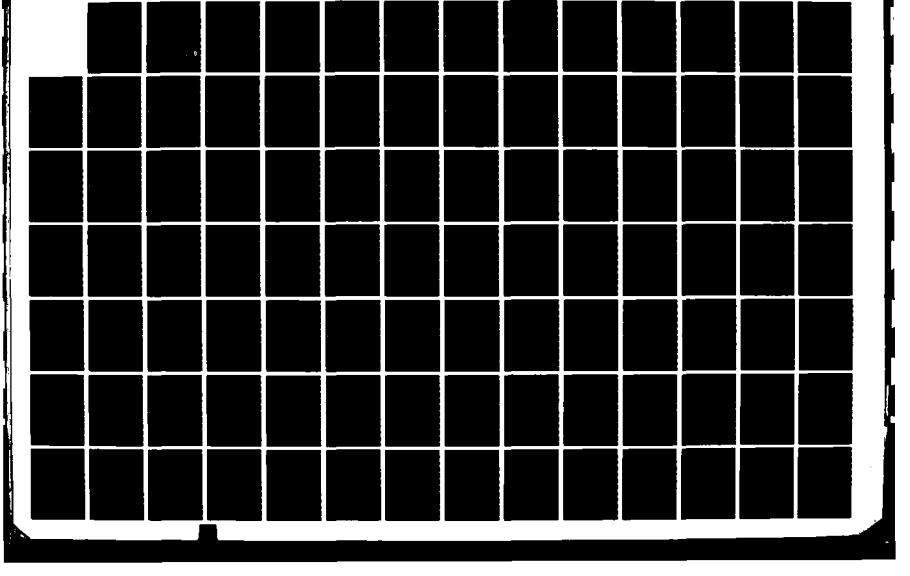
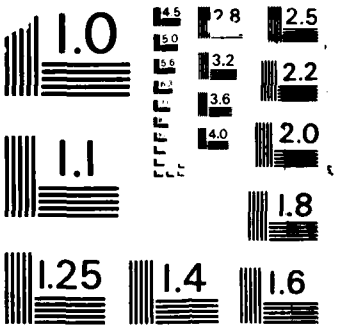
AD-A177 606 DESIGN OF A TCP (TRANSMISSION CONTROL PROTOCOL) AND IP (INTERNET PROTOCOL..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. N B METZEL

1/2

UNCLASSIFIED DEC 86 AFIT/GCE/ENG/86D-6 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

DESIGN OF A TCP AND IP SUBSET FOR INTEL 86/310 COMPUTERS

THESIS

Norman B. Hetzel
Second Lieutenant, USAF

AFIT/GCE/ENG/86D-6

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCE/ENG/86

DESIGN OF A TCP AND IP SUBSET

FOR INTEL 86/310 COMPUTERS

THESIS

Norman B. Hetzel
Second Lieutenant, USAF

AFIT/GCE/ENG/86D-6

AFIT/GCE/ENG/86D-6

DESIGN OF A TCP AND IP SUBSET FOR INTEL 86/310 RMX COMPUTERS

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Engineering

Norman B. Hetzel, B.S.

Second Lieutenant, USAF

December 1986

Acknowledgements

There are several people who I really need to thank for helping me get this put together in some semblance of order and put out. Many thanks goes to Major Garcia of the United States Army, my thesis advisor, for putting up with my coming apart at the seams at times, as well as Lt. Colonel Seward and Dr. Hartrum, my readers. Untold thanks goes to Captain Tim Weber and his wife Cheri for use of his computer when mine was occupied by my roommate and for patiently listening to and praying for me when I was in the middle of this. But the most thanks of all goes to the One who was alone responsible for my keeping my sanity, giving me what organization I had, and helping me to get this thesis out for His glory... my Lord and Savior, the living Jesus Christ. Believe me, He still does miracles!

Table of Contents

List of Figures

Abstract

As a precursor to a future gateway between the Air Force Institute of Technology (AFIT) Ethernet and the AFIT Computer Communications Laboratory Ethernet (CCLNet), the CCLNet was implemented using Intel Corporation iNA 961 and RMXNET software and then Transmission Control Protocol and Internet Protocol subsets were designed to be used on the Intel 86/310 RMX computer with OpenNET. Requirements definition, specification, design, and PL/M coding were used to solve the software design problem. The implementation and testing phases were not completed due to problems not covered in the system documentation.

I. Introduction

Problem Background and Motivation

Recently, the Air Force Institute of Technology (AFIT) has invested in Intel computers for research into Department of Defense and Army related applications. Two Intel 86/310 computers running the RMX operating system and one running the Xenix operating system were on hand, and more were to be ordered. As of December 1985, however, these computers were not being utilized to their full capacity by AFIT, since as stand-alone machines, their capabilities were limited in comparison to a network (18:11,12). These computers form a much more efficient and powerful resource by being networked together. AFIT has purchased a networking product by the Intel Corporation designed for use with these microcomputers. This local network is called the Computer Communications Laboratory Network (CCLNet). With the AFIT Transmission Control Protocol/ Internet Protocol (TCP/IP) Ethernet network interfaced with the system through a gateway as well, these computers form an even more important and very useful resource for the AFIT student. Moreover, this network is a valuable instructional tool for the networking class and a major resource for future theses in networking and related topic areas (18:12). The advantages to sharing resources in a network indicated that a network needed to be constructed (17:3-4).

Problem

As part of the gateway being constructed between the AFIT TCP/IP network and the CCLNet, the Intel iNA 961 and RMXNET software and Ethernet hardware needed to be installed and brought into operation and the Transmission Control Protocol and Internet Protocol software specifications needed to be adapted to the Intel 86/310 RMX computer and the software analyzed, designed, coded, and tested.

Scope

The overall AFIT TCP/IP Ethernet to CCLNet gateway project requires three separate software design efforts to be undertaken (see Figure 1). This project has concentrated on developing the TCP/IP software to later be used in developing both the datalink layer software interface to the AFIT TCP/IP network and the future gateway software converting from the TCP/IP to the Intel protocol. The CCLNet, the destination network, was brought up first to be used as a basis for building the TCP/IP software (see Figure 2) and to enhance familiarity with the Intel computers. The datalink layer of the Intel software is accessible, well documented, and was easily used; eventually, the TCP/IP software can be modified to fit the gateway datalink layer. This design effort was solely concerned with developing the TCP/IP software utilizing the datalink layer provided by the Intel Corporation networking software.

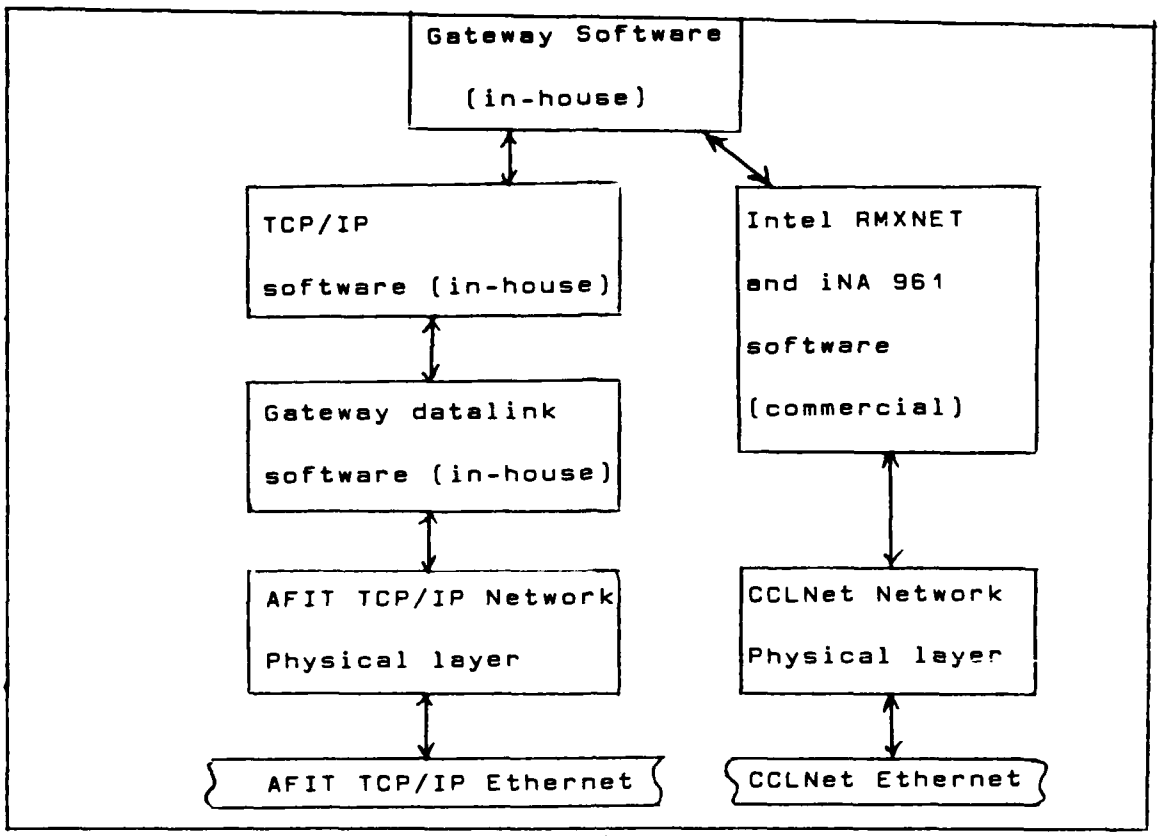


Figure 1. Overall Gateway Structure

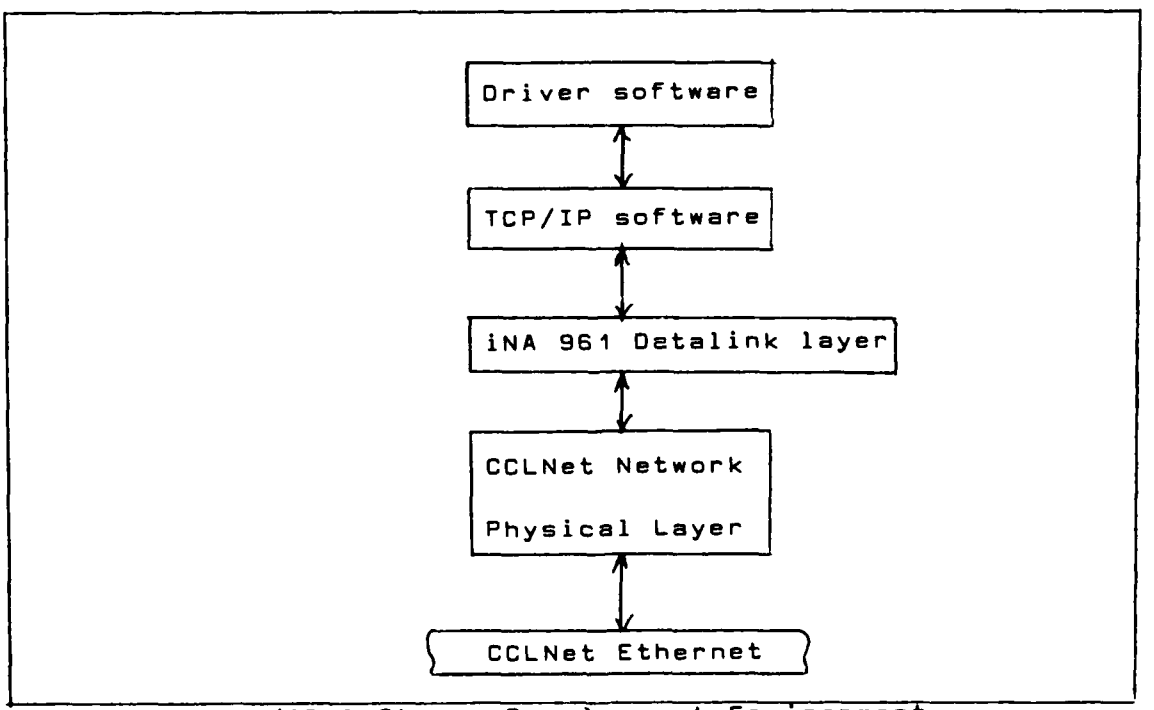


Figure 2. TCP/IP Software Development Environment

Assumptions.

The Intel networking protocol and the TCP and IP protocols are compatible enough at all levels to permit a gateway between the two to be constructed given the hardware interface.

Summary of Current Knowledge.

The reference manuals on the operation of the networking software and the interfaces to be worked with in designing the gateway modules are fairly complete and accurate, providing the necessary information to design the modules and the module interfaces. PL/M language compilers exist on the Intel computers; these were used to compile the module software which was written, edited, and subsequently debugged on the Intel systems.

Approach.

The problem was solved in two phases: installation of the CCLNet and the analysis, design, coding, and testing of the TCP and IP software on the CCLNet.

The implementation of the CCLNet was a necessary precursor to implementing the network interface. The CCLNet provided an operational network with an accessible datalink layer used by the TCP/IP software for debugging and testing purposes. The implementation of the CCLNet also provided a measure of familiarization with the Intel 86/310 RMX computers. The design involved in this implementation was

done by Captain Mark Weber (18). This phase of the project involved installation of the Ethernet cards and cabling, installation of the iNA 961 and RMXNET software from the Intel Corporation, generation of the RMX networking operating system from the installed software, and initialization of CCLNet operation.

The software modules were designed using several well-defined phases: requirements definition, specifications, design, coding, and final testing and evaluation. The requirements definition phase involved determining what functions were to be implemented and what specifically the software was to do. The specification phase involved the definition of the software module interfaces with the user and other software modules: the module inputs, the actions the module performs on the inputs, and the module outputs. The design methodology involved structure charts and flowcharts used in a top-down design approach. Coding was done in PL/M on the CCLNet.

Materials and Equipment

The equipment needed to implement the CCLNet on the two Intel 86/310 computers was an Ethernet cable, two Ethernet controller cards, two Ethernet cable transcievers, two connecting cables, and the iNA 961 and RMXNET software products from Intel. The software design and implementation required no additional materials.

Sequence of Presentation

Following this introduction, the TCP and IP models are presented, the Phase 1 CCLNet implementation is discussed, the requirements and specifications for the software modules are presented, the design of the software modules is presented and discussed, the testing and evaluation procedure is discussed, and conclusions drawn and recommendations given.

II. TCP/IP Background, Philosophy, and Functions

Background

The Transmission Control Protocol and the Internet Protocol are the Department of Defense standards and are not only very highly used, but are of the very few widely used networking protocols that are non-proprietary, and thus are well-documented in the available literature. These protocols were developed in order to build gateways from the ARPANET of the Department of Defense to other networks to facilitate the mutual sharing of resources (12:248). The philosophy and theory of the TCP and IP networking schemes will be briefly sketched in this section. For an in-depth exposition of the concepts described herein, the reader is directed to the TCP and IP references (14,15). First, the IP philosophy and purpose is discussed, followed by a short exposition of the major IP functions; then the TCP philosophy and purpose is discussed, followed by a short exposition of the TCP functions.

Internet Protocol

Internet Protocol Philosophy. The Internet Protocol was designed for use in gateway systems to move datagrams, packets of computer information, between different networks. The IP is a network-layer (level 3) protocol at a lower level than the TCP, being independent of the transport

layer and concerned only with taking the datagrams that it gets and moving them to their destination network and host computer (16:607). This protocol operates on a pure datagram basis, requiring the transport layer to perform the error-checking and datagram reassembly tasks. This protocol does not provide a reliable channel either end-to-end or hop-to-hop; there is no error control, flow control, retransmissions, or acknowledgements (14:3).

Internet Protocol Functions. The purpose of the IP is to move datagrams between networks. The design implements two basic functions: addressing and fragmentation (14:2). An IP module must exist on the gateway host to each network. These IP modules must contain compatible addressing and packet fragmentation and reassembly schemes. These modules may also have (depending upon the application) routing routines if they are needed (14:2).

The IP modules utilize the addresses to route datagrams through the gateways to the appropriate destination networks. Some gateways need this routing function; others that connect only two networks together do not (14:2).

At times, a datagram is routed through a network that requires the datagram to be fragmented to pass through due to operational constraints. The receiving IP module utilizes information placed in the internet header attached

to the fragmented datagrams by the sending IP module to reconstruct the original larger datagram (14:2).

Four major mechanisms are used by the IP module to implement these services: Type of Service, Time to Live, Options, and Header Checksum. The Type of Service mechanism involves the selection of parameters involved in servicing the datagrams according to whether the user is involved in interactive, bulk, or real time processing. The Time to Live mechanism prevents old packets that have somehow "gotten lost" from clogging up the lines. After the expiration of a time limit, the packet self-destructs. The Options mechanism allows the ambitious or special user to implement time stamps, error reports, and/or special routing. The Options mechanism isn't normally used with most common connections. The Header Checksum is computed for the header attached by the IP module only; the data inside the datagram still might contain errors that will not show up. If the header checksum does not verify, the datagram is immediately discarded without notifying the sending computer (14:3).

Transmission Control Protocol

Transmission Control Protocol Philosophy. The Transmission Control Protocol (TCP) is a transport layer (level 4) protocol providing reliable host-to-host communication (16:607). The TCP was designed by the

Department of Defense to be the standard for interprocess communication. It is end-to-end reliable and functions on a connection-oriented basis. It is designed to interface to a user process on one end and to the Internet Protocol or another protocol which performs a similar function (15:1).

The implementation of TCP will produce additional operating system calls producing the actions of OPENing or CLOSEing a connection through the network, SENDing or RECIEVEing data, and obtaining the STATUS of a connection (15:9). The communication between the application program and the TCP module, as well as the communication between the TCP module and the IP module, is assumed to be asynchronous (15:3).

Transmission Control Protocol Functions. The TCP module, in order to fulfill its purpose, needs facilities in six basic areas: basic data transfer, reliability, flow control, multiplexing, connections, and precedence and security.

In the area of basic data transfer, the TCP module needs to be "able to transfer a continuous stream of octets in each direction between its users by packaging some number of octets into segments for transmission through the internet system" (15:4). Blocks called letters can be submitted all at once for delivery by the TCP system, as well. As soon as the receiving TCP hits an end-of-letter

packet, the letter is then promptly delivered whole from the receiving TCP to the host (15:4).

In the area of reliability, the TCP module needs to be able to recover from data lost, damaged, duplicated, or delivered out of sequence by the system to provide a reliable system to the hosts. A sequence-number and positive acknowledgement system is used to facilitate this. If the ACK is not recieved within a timeout interval, the package is retransmitted and the process is repeated until it is successful. At the receiving end, a checksum is performed to confirm the integrity of the received packet; if this does not verify, then the packet is discarded and not acknowledged. The sequence numbers are used to correctly order the packets and to discard duplicates (15:4).

In the area of flow control, the "TCP provides a means for the receiver to govern the amount of data sent by the sender" (15:4). Every ACK packet contains a field indicating the acceptable sequence number range in which the receiving TCP will accept packets. This field either indicates the number of octets (in stream mode) or amount of buffer space consumed by the sent letters (in record mode) that can be transmitted through the network (15:5). The effect of this window changing would be to increase or decrease the rate of transmission and retransmission of

packets in accordance with the system load to provide greater efficiency (15:4).

In the area of multiplexing, several processes within a single host must be able to access the TCP independently of each other and at the same time. Each process is assigned a "port number" within that host, and this information is enclosed in the TCP datagram header (15:5).

In the area of connections, a three-way handshake mechanism is employed to reliably connect the two machines over the assumed unreliable underlying network. The three-way handshake used involves the opening machine sending a synchronization (SYN) packet, the listening machine sending back an acknowledgement (ACK) and a SYN in the same packet, and the opening machine replying with an ACK packet. After all of this has occurred without error, the connection is established and communication can proceed. The status of the ensuing data stream is maintained by the TCP modules until one side or the other terminates the connection, freeing the resources for other connections (15:5).

The TCP may or may not provide for security of varying levels. Such applications are not applicable in this instance, so the interested reader is referred to (15:5).

III. CCLNet Implementation

Before the gateway could be constructed, the network upon which the software development was to take place had to be implemented. This was Phase 1 of the project, the CCLNet Implementation. The goals and reasoning for the CCLNet will be discussed first, followed by the CCLNet design, the implementation procedure, problems encountered in the implementation, and conclusions and recommendations for the CCLNet.

Goals and Reasoning for CCLNet

The concept of the CCLNet to be installed in Room 245 of Building 640, AFIT, Wright-Patterson AFB, OH, stems from two needs: the need to utilize all equipment most efficiently and the need to provide a network to support classroom and support activities (18:11-12). The selection of the commercial product to be implemented followed from equipment on hand and the desire for extensibility (18:12).

Networks increase the utilization of resources. This follows from the fact that one processor cannot be constantly using its own memory, its secondary storage, computing, and using an I/O device all at the same time. If one connects several computers into the same network, it becomes possible for several computers to access the same memory banks, the same secondary storage devices, and the

same I/O devices, thus increasing the usage of all of the peripheral devices (17:3-4).

AFIT needed a working computer network to be accessible to students for both classroom and thesis work. The graduate-level introductory networking course in the AFIT Electrical and Computer Engineering Department did not have an easily accessible working network to illustrate the important OSI model concepts. Moreover, a network to support thesis efforts in this area was critical, as this was and is a rapidly developing area of study providing material for graduate-level theses.

The selection of the commercial product followed "from a convenient marriage between pragmatism and long-term objectives" (18:12). Already present in the Computer Communications Laboratory (Room 67 Building 640 AFIT Wright-Patterson AFB, Ohio) as of December 1985 were several Intel computers, acquired for other reasons and not being used. The choice of the Intel networking product made very efficient use of the materials already in the laboratory. Moreover, one of the long-term goals was to be both extendable and flexible to many vendors' products. The Intel OpenNET networking concept was designed towards that very goal (11:14). At present OpenNET is compatible not only with Intel's RMX and Xenix operating systems, but it is also compatible with Microsoft Networks (used to network MS-DOS products) and can also be used with IBM PC-DOS, as well

(11:14). In the future, the OpenNET will support products such as General Motor's Manufacturing Automation Protocol (11:14). The wide-ranging flexibility of this networking system combined with the resources already available at AFIT make the Intel system the obvious choice for the CCLNet.

Implementation Procedure

The implementation procedure followed for the software installations can be found in (7:Appendix A). The process involved eight major steps as follows: first, the iSXM 552 boards, the MOX 457 or 458 cables, and the Ethernet coaxial cable and transcievers needed to be installed. This hardware is used by both the OpenNET and TCP/IP subset software. Next, the software (both iNA 961 and RMXNET) needed to be installed from the floppies. This step corresponded with the coding and debugging of the TCP and IP subset software. After that, the Interactive Configuration Utility (ICU) needed to be run to locate the end address of the operation system without the RMXNET software. After the resultant files were modified to include the remote file driver, the SUBMIT file was run to finish the ICU process. The RMXNET system was then located in memory using the LOC86 utility, and the ICU was re-run to generate the total RMX and RMXNET operating system. The final SUBMIT file was run and then the system was booted.

IV. Requirements and Specifications

In order for a software development effort to be coherent, the desired end product must first be defined. The first two steps in the lifecycle process of most software products is the requirements definition and the product specifications, which effectively define the end product. The requirements for the software specify what the software is to do and what functions the software is to carry out. The main overall requirement of this thesis is that the software product, when completed, should facilitate communication between computers on the same Ethernet and that have TCP/IP subset modules. The specifications for the module software involve desired inputs, outputs, and module interfaces. This section covers the specific module requirements and specifications for both the IP and the TCP subsets.

Internet Protocol Requirements and Specifications

IP Requirements. The Internet Protocol subset module provides a datagram service for use by the Transmission Control Protocol subset module, utilizes the existing OpenNET data-link software and physical layer hardware of the CCLNet, provides addressing headers for use by the future gateway in routing the packets to the proper destinations, provides the framework for modification into a

full Internet Protocol by addition of fragmentation and options software (14:2-3).

A datagram service as required here indicates that the IP subset module is to send and receive each packet as an individual unit. The IP subset module must attempt to deliver the packets presented to it, but it is not required to ensure error-free reception in order of transmission. The IP does not send acknowledgements and does not use sequence numbers to order the packets. A packet identification number is used in the fragmentation process; if the packet is too large to fit through the route to the destination machine, the packet must be fragmented and reassembled by the IP modules involved (17:188-189).

The IP subset software should utilize any existing software and hardware resources available to facilitate efficient resource utilization. The Intel OpenNET software and hardware existing on the target machines has a user-accessible data link layer coded to IEEE standard 802.3 (5:Chapter 3,1). Since this standard applies to all Ethernet networks (10:134), this networking software and hardware should be compatible with other TCP/IP Ethernets. Therefore, to minimize the time requirement of this project, the Intel OpenNET datalink software and hardware was utilized in the IP software development effort.

Communication of certain information between the two IP subset modules in a connection is a requirement. The method

used by the full Internet Protocol is the IP header, attached to the outgoing packet by the sending IP and removed by the receiving IP. In order to facilitate upgrading to full IP status, the standard IP header was used to implement this function [14:7].

The IP software is designed to be the basis for a future gateway between the CCLNet and the future AFIT TCP/IP Ethernet. The need, however, was to design and construct a subset for use on the CCLNet as a precursor to the gateway project. As a result, since the IP module is not going to be communicating with any other IP modules outside of the CCLNet, packets do not need to be fragmented to pass through the system. As this software will eventually be used to produce this gateway, addition of this function should be made as easy as possible. Since this software is not an immediate concern, it will be left to future efforts to design and implement.

The options field of the header provides for the routing functions, for additional timestamping of packets between networks, and for security information [14:16]. The routing function is not of immediate concern, as this preliminary subset is being designed for use on the CCLNet only. The function of this IP software as a gateway necessitates the easy addition of the routing function through the options field.

The security and the timestamping are also issues that are taken care of within the local network by the TCP module. Again, the upgraded software needs to have these functions implemented in order to be compatible with other IP modules; therefore, this implementation provides for the easy addition of these functions.

IP Specifications. The Internet Protocol subset module communicates with using processes through system calls, with the remote IP subset module through the IP Header, and with the datalink layer of the OpenNET software through a procedure call and request blocks. The IP module provides two functions to the user: IPSEND and IPRECV. The header used by the IP module, shown in Figure 3, contains the following fields: header version, internet header length (IHL), Type of Service, total length, identification, fragmentation flags, fragment offset, time to live,

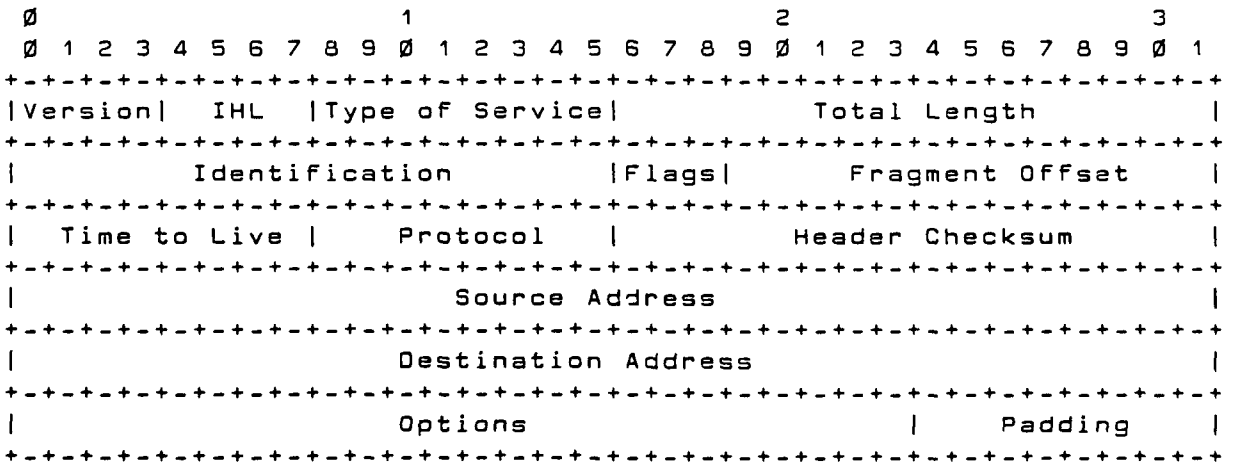


Figure 3. Example Internet Datagram Header. (14:11)

protocol, header checksum, source address, destination address, options, and padding if needed. The procedure call used to interface with the datalink software is CQ$COMM$RB, and the two request blocks needed are TRANSMIT and POST$RBD.

The two calls added to the operating system by the IP module are the IPSEND and the IPRECV (RECeiVe) calls. The SEND call takes the arguments supplied in the call, constructs a header from these and the assumed values for the other fields (covered in the discussion of the header fields later on in this section), attaches it to the datagram, and sends the datagram to the address specified. If an error occurs in transmission or header construction, the call returns "Error" in the result field; if the acknowledgement for the packet sent is received, the call returns "OK" in the result field (14:35). One of three things can happen once the datagram is successfully sent: the datagram can find the appropriate process in the destination host and find a pending RECV call, it can find the appropriate process in the destination host and not find a pending RECV call and thus initiate a "datagram pending" message to the receiving process, which then acts, or it can find the host but not find the correct process, in which case the host discards the datagram and sends an "error" packet back to the source IP subset module. If the packet encounters a pending RECV, the call is satisfied by passing data from the IP module to the user (14:36). The format of

these two system calls is as follows: SEND(dest, TOS, TTL, BufPTR, len, Id, DF, options => result); RECV(BufPTR => result, source, dest, prot, TOS, len). The abbreviations are given in Table 1. The result parameter returns either an OK or an ERROR value. The other parameters are either self-explanatory or correspond to header fields discussed below.

TABLE I (14:35)

Parameter Abbreviations

Parameter	Contents
dest	destination address
TOS	type of service
TTL	time to live
BufPTR	buffer pointer
len	buffer length
Id	identifier
DF	don't fragment flag
options	option data
result	response:
source	source address
prot	protocol (14:35)

The IP module header has five 32-bit words in the minimum configuration. The first two fields of this header contain the version and the internet header length. The IP

has gone through several versions to date; the version described herein and in the reference [14] is version number 4. The internet header length contains a four-bit number giving the length of the header in 32-bit words. As was mentioned, the value in this field will be five.

The type of service field contains an eight-bit number that provides information on the characteristics of the IP operation. The first three bits of the field give the precedence of the service desired as compared to other IP operations on different TCP's either on the same host or a different one. This is a priority variable not used in this implementation; the higher the value in the precedence variable, the more important the communication is and the more direct attention given to these particular incoming and outgoing datagrams. The next bit is a flag indicating whether the service is stream or datagram, telling the IP module whether to expect to send out batches of datagrams created from letters or to send out a constant stream of datagrams (perhaps an interactive session, for example.) The next two bits give the relative reliability level desired running from ØØ being the lowest reliability to 11 being the highest. The next bit indicates which of either speed or reliability is more important in the IP module performance. The last bit selects one of two speeds: high or low [14:12].

The next two fields in the IP header contains the total length and identification fields. The total length field is sixteen bits long and gives the total length of the entire datagram, header and data, in octets. The identification block is also 16 bits long and gives the receiving IP subset module help in assembling packet fragments (14:13).

The fragmentation flags take up three bits. The first bit is reserved and is always set to zero. The second bit is the Don't Fragment flag, set when the datagram is not to be fragmented (a value passed in by the system call argument) either by this IP or by any other IP enroute to the destination address. The third bit is the More Fragments bit, used by the fragmentation algorithm to indicate to the IP that more fragments of the same original datagram are contained in ensuing packets (14:13).

The fragment offset field is also used with the fragmentation software. This field measures in octets the offset from the beginning of the original datagram to the position of the contents of this individual packet within that datagram. The first fragment in the datagram has a zero offset (14:13).

The time to live field measures in seconds the total maximum lifetime of the packet. If the packet does not find its destination by the time that this field is decremented to zero (by intervening IP's), it self-destructs, preventing it from clogging the lines (14:13).

The protocol field is the field specifying what upper level protocol is being used in the connection specified, so that the IP module knows which module to pass the datagram onto. The only protocol that this implementation will interface with is TCP; therefore, according to reference (13:6), the number in this field will be six.

The next field contains the header checksum. This is the one's complement of the sixteen-bit sum of all of the sixteen-bit blocks of the header. Each time an intervening IP changes the header (for example, decrements the time-to-live value), it recomputes the checksum of the header only (with zero's substituted into the header checksum field) and updates the header checksum field accordingly (14:14,29).

The source and destination address fields contain the coded source and destination addresses according to the selected network addressing scheme.

The options are miscellaneous items that might be useful to some users such as time stamps, security information, routing information, or other miscellaneous information (14:16-21). In the IP subset module, the option field will contain one octet of zeros, as none of the options apply within the scope of this thesis. The options may be added to this IP implementation as part of a later, follow-on effort.

The interface with the Datalink layer of the Intel OpenNET software will be through the system call

CQ$COMM$RB(reqblktoken, except$ptr) where req$blk$token is the iRMX token for a request block segment, defined below, and a pointer to a word containing the exception code returned by the call (5:Chapter 2,3;6:Chapter 3,1). The request block is a segment of memory allocated by the software from the pool of available memory and structured to contain the information needed by the data-link layer to perform its job.

The structure of the request block passed to the system for the TRANSMIT command, which transmits a packet over the Ethernet, is given in Figure 4 (next page). The reserved fields are not set by the IP subset module, as they are used by the iNA 961 software. (6:Chapter 3,1) The length is the total length in bytes of the request block memory segment. The user is a field set to Ø in this version of iNA 961 (6:Chapter 3,1). The response_port field is to be set to ØFFH (6:Chapter 3,1). The device_id and port_id fields are to be set to the device and port on that device that the request block is created by and returned to (6:Chapter 3,2). The subsystem field is to be set to 2ØH, indicating the datalink layer software is to process the block (6:Chapter 3,2). The opcode is specific to the command used; in this case, the opcode is 84H or the TRANSMIT command (5:Chapter 3,8). The response is a condition code returned by the iNA 961 software indicating the results of the block processing. These codes can be found in

```
DECLARE RB structure (
 reserved_1     (2) word,
 length         byte,
 user           word,
 response_port  byte,
 device_id      byte,
 port_id        byte,
 reserved_2     (2) byte,
 subsystem      byte,
 opcode         byte,
 response       word,
 reserved_3     word,
 buf_count      word,
 byte_count     (4) word,
 buf_loc        (4) dword,
 dst_addr_ptr   dword );
```

Figure 4. TRANSMIT Request Block PL/M Declaration

(5:Chapter 3,8;6:Chapter 3,1)

(5:Chapter 3,22). The buf_count parameter is an integer from 1 to 4 indicating how many buffers of information exist to be sent (5:Chapter 3,8). The byte_count array contains the length in bytes of each buffer (5:Chapter 3,8) and the the buf_loc array contains the absolute

addresses of the start of each of these buffers (5:Chapter 3,8;6:Chapter 3,2). The dst_addr_ptr is the absolute address of the destination computer (5:Chapter 3,8;6:Chapter 3,2).

The other type of request block to be used by the IP subset module to communicate with the datalink software is the POST$RBD block (given in Figure 5), which sets aside

```
DECLARE RB structure (
 reserved_1     (2) word,
 length         byte,
 user           word,
 response_port  byte,
 device_id      byte,
 port_id        byte,
 reserved_2     (2) byte,
 subsystem      byte,
 opcode         byte,
 response       word,
 disp           byte,
 reserved_3     word,
 byte_count     word,
 buf_loc        dword );
```

Figure 5. POST$RBD Request Block PL/M Declaration (5:Chapter 3,12;6:Chapter 3,1)

buffers for the OpenNET software to place the incoming packets. The only differences between this block and the TRANSMIT block are that the opcode is 86H (13:Chapter 3,12) and the dlsap variable is a code for the process that initiates the call (5:Chapter 3,2-3).

The buffers passed to the iNA 961 software need to have some preliminary information attached to them, as well. Up to four buffers may be transmitted by the TRANSMIT command; the format for the first of these is given in Figure 6. The other buffers contain only data. The destination_lsap

```
DECLARE buffer structure (
 destination_lsap  byte,
 source_lsap       byte,
 iso_cmd           byte,
 data              (*) byte);
```

Figure 6. Buffer Structure PL/M Declaration (5:Chapter 3,8)

number is the number representing the destination process to which the packet will be delivered (5:Chapter 3,2,9). The source_lsap is the number for the initiating process (5:Chapter 3,2,9). The iso_cmd is to be set to 03H as required by IEEE 802.3 (5:Chapter 3,9). The buffers returned by the POST$RBD command also have this structure

plus initial fields for destination_addr, a sixteen byte array containing the host id of the destination node, source_addr, a sixteen byte array containing the host id of the source node, and information_len, a word containing the number of information bytes including TCP and IP headers in the received packet (5:Chapter 3,11-12).

TCP Requirements and Specifications.

TCP Requirements. The Transmission Control Protocol subset module is to provide a "... reliable, securable logical circuit or connection service between pairs of processes" (15:3). In order to accomplish this goal, this module needs to address the problems of data transfer, reliability, flow control, and connection management (15:3). The problems of multiplexing connections and precedence and security of communications do not need to be addressed (15:3).

The problem of data transfer, communicating submitted buffers to the specified remote station, needs to be addressed by the Transmission Control Protocol subset module. The user should be able to make a system call supplying the data buffer and pertinent information (given in the TCP Specifications section) and be advised by a returned condition flag whether or not the information was received at the desired remote destination (15:4).

The issue of reliability involves ensuring reception by the destination process of the data sent in the condition and order that it was given to the sending TCP subset module. The sending TCP subset module should be able to ascertain whether or not the data sent has been correctly received by the remote process and should relay this information to the using process (15:4).

The term flow control refers to remote control of the packet transmission rate of the TCP subset module. If the receiving TCP slows down due to an uncontrollable factor such as heavy time-sharing use, the sending TCP subset module needs to be able to slow down the rate of transmission in response so that the number of lost packets is minimized. If the remote TCP wants to increase the transmission rate, the sending TCP subset module needs to be able to adjust (15:4).

The TCP subset module needs to be able to manage the connections between itself and other TCP's. Connection management involves keeping track of vital statistics of the data connection during the time when the connection is active (15:48) and managing the opening and closing handshakes. The data statistics being kept should be accessible by a system call, returning all of the pertinent data to the screen of the using processes' terminal. The opening and closing handshakes should also be initiated by system calls (15:5).

The multiplexing facility and the precedence and security facility are not needed for this project. Since this subset is a preliminary implementation, its main concern is only to connect a single process with another. Precedence and security are useful features in operational systems; since, however, these features are not directly connected with the problem of data communication and management addressed in this project, these are excluded. Both of these facilities can be added at a later date after the preliminary TCP subset modules are operational (15:3).

Module Specification--Transmission Control Protocol Module. The TCP subset module will interface with the user by system calls and will interface with the lower level software through procedure calls (defined in the IP subset module specifications) and a packet header. The TCP module provides six functions to the user: open, send, receive, close, status, and abort. The TCP header contains the following information fields: source port, destination port, sequence number, acknowledgement number, data offset, flags, window, shecksum, urgent pointer, and options. In addition, the TCP subset module uses the IA$ADD and MC$ADD OpenNET datalink calls to initialize the OpenNET datalink software to be used by the IP subset module.

The OPEN call initiates the handshaking routines of the TCP subset module and returns a local connection name which, during the lifetime of the connection, refers to the local

socket-foreign socket pair. The format of the OPEN statement is as follows: OPEN (local port, foreign socket, active/passive [,buffer size] [,timeout] [,precedence] [,security/compartment])-->local connection name. The local port variable contains the binary code representing the local process within the local host. The foreign socket information variable contains code for the destination port (process) and the destination address (host). The port is used in the TCP header construction; the address of the host is needed for the IP module call after the TCP datagram is produced. When the active/passive flag is set to active, the TCP starts the connection procedures by sending the appropriate packets (see design section of this paper). The passive option turns this call into a LISTEN for connection requests from remote processes. The BUFFER SIZE, TIMEOUT, and SECURITY/COMPARTMENT options are specified in the references (15:43,44), but implementation of these options is left for future thesis efforts (15:43-44). Buffer size and timeout are set to default parameters by the software; the option of letting the user set these is a convenience, but is not necessary to implement data communication and management. The security/compartment option falls under the precedence and security facility, not implemented here as stated in the TCP Subset Module Requirements section above.

The SEND call transmits the contents of the indicated buffer through the connection. The local connection name is

the same value returned by the OPEN command used to establish the connection. If an OPEN command has not been given for the connection name given, the TCP considers the call an error and ignores it. If the EOL flag is set, the buffer contains the last packet in a letter, and the EOL flag is set in the packet formed. The SEND statement format is as follows: SEND(local connection name, buffer address, byte count, EOL flag, URGENT flag [, timeout]). If an OPEN command has not been given for the connection name given, the TCP considers the call an error and ignores it. The URGENT and the TIMEOUT parameters are specified in the reference (15:45), but will not be implemented. As before, the timeout is set by default; the urgent flag falls under the category of security and precedence. For purposes of this project, these parameters are ignored, but could be implemented as part of a future project (15:45).

The RECEIVE call sets aside a buffer of specified (byte count) size into which the TCP subset module will place incoming letters, and if a buffer is full of data, it will be returned. If an OPEN statement has not initialized the connection before the call is made, an error message will be produced. The buffer will be filled with either the entire letter received or as much data as the buffer can hold. In the first case, a true END_OF_LETTER flag will be returned along with the buffer to indicate that the letter is finished. In the second case, no EOL indication will be

returned. The RECEIVE statement format is as follows: RECEIVE(local connection name, buffer address, byte count,result) where the parameters are as defined above (15:46-47).

The CLOSE statement initiates the handshaking sequence terminating the connection. The use of a CLOSE system call indicates that the calling process has no more data to send, but will accept incoming data until the remote process also issues a CLOSE. At that time, the connection is terminated. The CLOSE statement format is CLOSE(local connection name) (15:47-48).

The STATUS command will produce a data screen containing information on the local socket, the foreign socket, the local connection name, the receive window, the send window, the connection state, the number of buffers awaiting acknowledgement, the number of buffers pending receipt (including partial buffers), the size of the receive buffers, the urgent state, the precedence, the security/compartment, and the default transmission timeout. The options listed above not implemented in this version of the TCP (urgent state, security/compartment) will output a null entry in the output screen. The STATUS statement format is STATUS(local connection name) (15:49).

The ABORT system call simply causes all outstanding SENDs and RECEIVEs to be dropped and a packet with a true RST (reset) flag sent. The user will receive an

acknowledgement of the ABORT call. The ABORT statement format is ABORT(local connection name) (15:49).

The TCP header is the method by which the two networking software modules on each end of the connection communicate. The first two fields in the TCP header, shown in Figure 7, contain the source port and the destination port names. The sequence number field, thirty-two bits long, contains the number used by the remote machine to reassemble the packets and to acknowledge. The acknowledgement number field contains the sequence number of the last packet data octet received by the TCP subset module. The data offset field contains the number of thirty-two bit words in the TCP header. The TCP header, including options, will always be padded out to an integral multiple

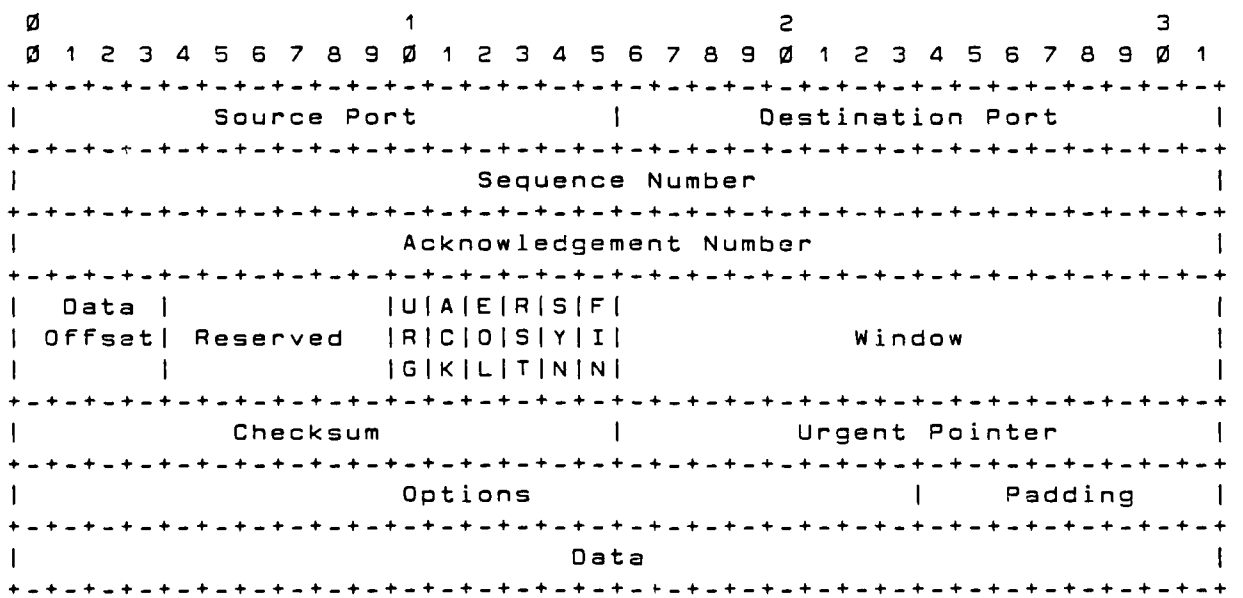


Figure 7. TCP Header Format (15:15).

of thirty-two bits (15:19). The "reserved" field is set to zero. The control bits (URG, ACK, EOL, RST, SYN, FIN) indicate special packets. A high URG bit says that the urgent pointer field is significant. In this project, this will not happen. A high ACK bit indicates an acknowledgement; A high EOL bit indicates end-of-letter. A high RST bit tells the receiving TCP to reset the connection. A high SYN bit is used to synchronize the initial sequence numbers of the two TCP's. A high FIN bit is used to tell the receiving TCP that the sending TCP has no more data to send. The window field indicates the number of data octets starting with the sequence number in the acknowledgement field that the receiving TCP will accept. The checksum field contains the one's complement of the sixteen-bit sum of the header broken down into sixteen-bit words. The checksum field, for purposes of the computation, is replaced with zeros. The urgent field and the options are defined by the reference (15:17-19), but will be left to future projects to implement (15:15-19).

The setup calls for the OpenNET datalink software utilize the same procedure call that the IP subset module used to transmit and receive buffers: CQ$COMM$RB (see IP specifications.) The type of request block to be used by the TCP module to interface with the datalink software to set the host machine's Ethernet identification is the IA$SETUP block, given in Figure 8 (next page). The only

```
DECLARE RB structure (
 reserved_1     (2) word,
 length         byte,
 user           word,
 response_port  byte,
 device_id      byte,
 port_id        byte,
 reserved_2     (2) byte,
 subsystem      byte,
 opcode         byte,
 response       word,
 reserved_3     word,
 count          word,
 address        (6) byte);
```

Figure 8. IA$SETUP Request Block PL/M Declaration (5:Chapter 3,14;6:Chapter 3,1)

structural differences from the TRANSMIT block (see IP specifications) are that the opcode is 89H (5:Chapter 3,14), the count is a variable specifying the number of bytes in the host ID (this number is required to be six), and the address is a six-byte array containing the host node Ethernet address. (5:Chapter 3,14).

The other request block needed to initialize the datalink software is the MC_ADD block (Figure 9, next page),

```
DECLARE RB structure (
  reserved_1     (2) word,
  length         byte,
  user           word,
  response_port  byte,
  device_id      byte,
  port_id        byte,
  reserved_2     (2) byte,
  subsystem      byte,
  opcode         byte,
  response       word,
  reserved_3     word,
  count          word,
  mc_address     (6) byte);
```

Figure 9. MC$ADD Block PL/M Declaration (5:Chapter 3,12;6:Chapter 3,1)

which adds a remote address to the local datalink software broadcast list. The only structural differences from the TRANSMIT block (see IP specifications) are that the opcode is 87H (5:Chapter 3,15), the count is the number of bytes in the multicast address (required to be six), and the mc_address variable is an array of six bytes containing the Ethernet address of the destination node that this TCP/IP is to communicate with. (5:Chapter 3,15).

V. Software Design Discussion

The design of a set of software modules should take into account each area of the software: structure, control flow, data structure, algorithms, and requirement and specification compliance. The designs of the TCP subset and the IP subset modules in these areas are discussed below.

IP Subset Design.

The software was structured into two procedure bodies, as the functions performed by each system call are easily met by this structure. The control for both procedures comes from the user process, which passes the control of the processor to the IP subset procedures and receives control back at the conclusion of the packet processing. The data structures used to communicate with the user process are the parameters passed in with the call. The request block is used to communicate with the lower level software (see specifications). The buffer pointers are stored between reception by the lower level software and processing by the IPRECV function in the buffers passed to the datalink software.

The software was designed to meet the requirements and specifications presented above. This software provides an unreliable datagram service to the user process, in this case the TCP, it utilizes the OpenNET Data-link software and

physical layer hardware as the interface to the test network, it utilizes the IP header presented in the specifications along with a header checksum to provide the addressing information necessary for a gateway implementation, and it provides the commenting to support the future addition of fragmentation and options software as included in the reference.

This design provides the required datagram service to the user process. The IP takes the packet given to it, wraps it in the proper header, sends it over the network, takes the packet at the other end, unwraps the header, and gives the packet to the destination host. The only error checking performed is a header checksum on the IP header itself-- no checking is performed on the data. If a packet has a header with an error, it is simply dropped without notification. This design provides a connection that attempts to send what it gets and delivers what it receives to the user process, but does not ensure order, reception, or quality of reception.

The Intel OpenNET software and hardware is utilized through the interface defined in the specifications. This hardware and software provide the data-link and physical layers of the networking software. Instead of writing the lower layers of the software, the use of this software allowed instead the writing of software tc create the proper request block in memory and the proper function calls to

send and receive the desired packet, reducing the work needed to implement a basic communication between machines. The iNA 961 software and the computer hardware take care of the rest of the sending job.

The header implemented is the standard header given in (14:11), providing the necessary information for the receiving IP module and the receiving process. Each field given is filled in from the information given according to the specification, the sum of all of the 16-bit words taken modulo 16, and then the one's complement of that taken. This is subsequently placed in the checksum field. The address fields and the future inclusion of the options field provide the addresses and information needed for the routing function of the IP module.

The features to be added in later, the fragmentation and the options software, are clearly marked as to where they occur in the code. These features can be implemented as procedure calls from the indicated spots in the code, or they can be added to the code in the spots indicated directly.

TCP Subset Design.

The areas addressed in the Transmission control protocol subset design break down into four broad categories: TCP subset structure, control flow, data flow, and algorithms. Moreover, the design of the subset module

was directly connected to the requirements of data transfer, reliability, flow control, and connection management.

The TCP subset is broken down into two modules: the TCPMODULE software and the TCPKERNEL software. The former contains the eight user-TCP interface procedures. The latter contains the TCP itself, including all of the various procedures for each of the possible states a connection could be in and GETPACKET and SENDPACKET procedures, as well. The system is split up into two modules to prevent the processing of a local system call from keeping the software from processing an incoming packet from the remote system. The natural split would therefore occur between the user calls and the rest of the TCP software, since the TCP subset would be its own task, and the interaction procedures would be under the user's task (2:27-29). The interface between the two modules is to be in the form of public data structures known to both software modules (2:233-234).

The design of the TCP subset incorporated two separate control flows to operate in the timesharing environment of the RMX system (9:Chapter 5,4) to facilitate independent user calls and incoming and outgoing packet processing (see above). The kernel module must be continuously polling to process incoming packets and send outgoing packets independently of the user system calls.

In the TCPKERNEL module, the actions performed are dependent upon the value of a public variable STATE,

indicating the state of opening or closing of the connection (see figure 10, next page). The states on the chart are as follows: The CLOSED state represents no connection existing. The LISTEN state represents the TCP subset waiting for a remote SYN packet from another TCP. The SYN-SENT state represents the TCP subset waiting for an ACK of a SYN it sent to open the connection. The SYN-RECEIVED state represents the state between the reception of the remote SYN and the reception of the ACK to the local SYN packet. The ESTABLISHED state represents a fully open and operational connection. The FIN-WAIT-1 state represents a local FIN packet sent and needing an ACK as well as waiting for a remote FIN packet. The FIN-WAIT-2 state represents the time waiting for the remote FIN packet. The TIME-WAIT state represents the waiting state while the network delivers the final ACK of the remote FIN to the remote host. The CLOSE-WAIT state represents waiting for a CLOSE function call from the local user after a remote FIN has been processed. The CLOSING state represents waiting for the ACK to the local FIN sent in reply to the remote FIN. (15:21) The state that the connection is in is recorded in the variable STATE, which is a public variable modified by the interface procedures and the kernel procedures depending upon the event processed (15:58-73).

The system interface procedures, on the other hand, enter into the control flow of the user process. The user

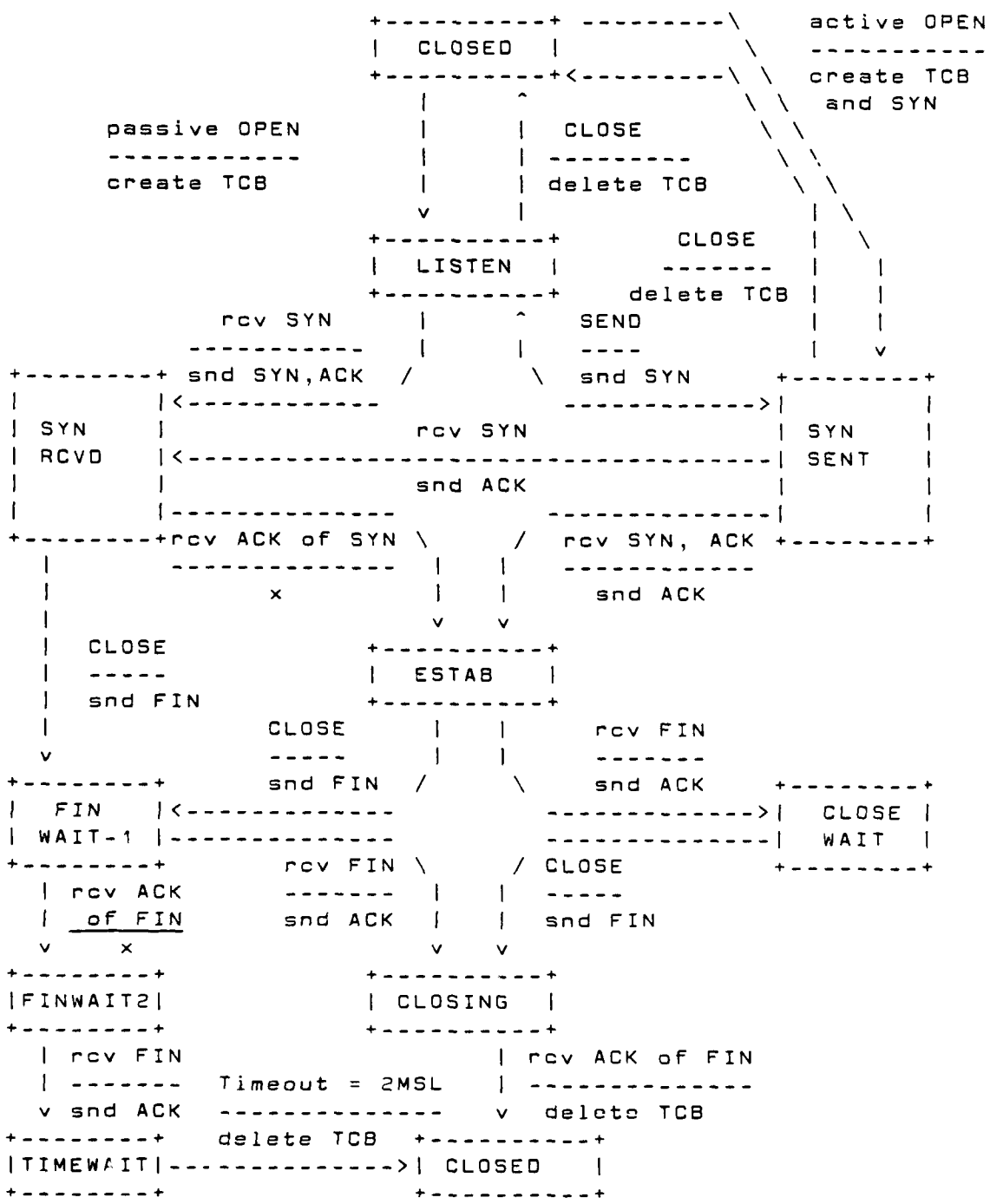


Figure 10. TCP State Diagram [15:23].

process hands control to the interface procedure, which then uses the call parameters to modify the appropriate public variables. Control is then handed back to the user process while the information in the public variables is concurrently acted upon by the kernel in accordance to whatever state the connection between the computers is in.

The public variables of the TCP module subset are of central importance, since they form the interface between the system call procedures and the TCP subset kernel program. The variables are of two types: the Transmission Control Block (TCB) variables and the service queues.

The Transmission Control Block variables are those individual variables used by the TCP subset to manage the connection and used by the system interface procedures to affect the actions of the kernel module. They are given in Table 2 (next page). These variables are accessible by both the procedures in the TCPKERNEL module and the system interface procedures (15:19-21,48).

The other data structure used is the queue. Three queues are utilized by the TCP subset modules: the first is the send queue of maximum size 100 containing fields in each queue entry for local connection name, data buffer pointer, data buffer byte count, and the END_OF_LETTER flag. The second is the receive queue of maximum size 100 having fields in each queue entry for the local connection name, data buffer received, and the data byte count. The third

TABLV II

Transmission Control Block Variables

Variable name	Description	(PL/M type)
LOCALCONN	Local connection name	(word)
FOR$SOCKET	Foreign socket name	(word)
LOC$PORT	Local port name	(word)
STATE	Connection state	(byte)
SNDSEQ,SNDACK, SNDNXT,SNDLBB, SNDUNA,SNDWND, SNDWL	Parameters affecting outbound segment	(dword) (11:8Ø)
CTLACK,CTLRST, CTLSYN,CTLFIN, CTLEOL	Flags for outbound packet header	(byte)
SEGSEQ,SEGLEN, SEGACK,SEGWND, SEGUNA	Parameters of last received remote packet	(dword)
RST,ACK,SYN, FIN,EOL	Flags from last received packet	(byte)
RCVNXT,RCVLBB, RCVWND,RCVBS	Processing info. for incoming segments.	(dword)
IRS	Initial receive sequence number.	(dword)
ISS	Initial send sequence number.	(dword).

queue is the retransmit queue of maximum size 100 containing fields in each queue entry for local connection name, data buffer pointer, data buffer byte count, END_OF_LETTER flag, packet timestamp, and packet sequence number. (15:45-46,73).

The algorithms used within the various state subroutines were given by the major reference.(15:58-73) These algorithms constitute the Transmission Control Protocol and what should happen for a connection in each state for each incoming packet condition.

The TCP subset was designed to fulfill the four basic requirements as given above: provide for data transfer from one host to another, provide for reliability, provide for flow control, and provide for connection management.

The first requirement, data transfer, is provided by the SENDPACKET and GETPACKET routines and the RECEIVEQ queue. SENDPACKET takes the first packet in the send queue, places a header on it from the information in the TCB, and gives it to the IP subset module. GETPACKET, if any packets are ready, takes the packet returned by the IP subset module and then removes the header, places the information thus attained into the TCB, and places the buffer pointer and byte count into a small data structure called INPUT$PACKET. The TCPKERNEL program then proceeds to process the packet and transfer the buffer pointer and the byte count to the receive queue, where the user can access it through the RECEIVE system call. If the packet processed through all of

the reliability checks sucessfully, a TRUE value is returned in RESULT; otherwise, a FALSE value is returned.

The second requirement, reliability, is provided for by several mechanisms: sequence numbers, checksums, acknowledging, and the three-way handshake initiation algorithm. Sequence numbers, numbering each packet as it is created at the sender, are used to ensure that the received packets are delivered in order to the destination process (15:4). Checksums provide an insurance against internal errors in the received packet due to network or host hardware problems. (15:4,17:125,130) The checksum (the one's complement of the modulo-16 binary sum of the packet) is computed by the sender and then recomputed by the receiver. If the two do not match, the packet is discarded at the receiver and is retransmitted when the timestamp expires. The acknowledging mechanism provides the information to the sending computer that the receiving computer has received the packet in correct form (15:4). The reliability of the initial connection is ensured by the three-way handshake system that requires both computers to exchange SYN packets and to acknowledge each other's SYN packet before allowing the properly established communication to ensue (15:5).

The third requirement, flow control, is provided by the window mechanism (15:4). If the receiver wishes to change the data rate, the window field of the packet sent by the receiver can be either enlarged (to speed up the rate of

transmission) or reduced (to slow down the rate of transmission.)

The fourth requirement, connection management, is provided using the Transmission Control Block variables. (15:5) Again, the handshake mechanism (also mentioned under the reliability heading) sets up the connection using the TCB variables and keeps track of it, again using the TCB variables.

VI. Results

CCLNet Implementation Phase

The CCLNet Implementation phase of the project has been completed. The major obstacles encountered in this phase of the project were the acquiring of hardware and the learning curve associated with using the Intel systems.

The hardware problem centered around missing Ethernet transceivers. These had been ordered, but did not come in from the original ordering company. They had to be reordered from a different contractor. In the interim, a Digital Equipment Corporation DELNI Ethernet-in-a-box was borrowed from a faculty instructor to facilitate network construction. The transceivers were eventually received and installed.

Software Development Phase

Requirements Analysis. The requirements analysis phase proceeded without problems. The requirements for the TCP and IP subsets were modified from the requirements and specifications given in [14] and [15] and are presented in section 4 of this thesis.

Specifications Review. The software specifications review proceeded without problems. The specifications for the TCP and IP subsets were drawn from those specifications

given in (14) and (15) and are presented in section 4 of this thesis.

Software Design. The software design phase proceeded without problems. Structure charts and flowcharts (given in the Appendix) were used in a top-down design approach. The discussion of what features were included in the design and why is given in section 5 of this thesis.

Software Coding and Syntax Debugging. The software coding and syntax debugging ran into a few problems before completion. The software, originally to be coded into C, was rearranged into PL/M when the operating system would not produce working operating system calls from C programs. The documentation of the RMX operating system is written with examples in PL/M, which has totally different call parameter conventions than does C (5,6,7,8,9). The differences in the manner of calling were great enough that, in order to facilitate easier and faster coding, the coding was restarted and completed in PL/M. Once this change was made, the coding and syntax debugging proceeded without problems.

Software Implementation and Semantic Debugging. This phase of the project ran into some problems. The major problem that was not solved involved communication between the developed software and the Intel iNA961 software datalink layer. Simply put, the author could not get the datalink software to process a request block submitted by a user program. A small test program was written to fill in

the fields of the IA$SETUP request block and submit it to the iNA 961 datalink software. As noted in Figure 8, the request block requires PORT_ID and DEVICE_ID fields to be filled in. The documentation adequately covers the information needed to fill in the other fields, but does not cover the information needed to fill in these two fields (5,6). Experimentation and exhaustive research failed to turn up a working value for PORT_ID, though the value of DEVICE_ID was finally discovered. As a result, no semantic or logic debugging took place, since the test could not be run on the required datalink layer.

Software Testing. This phase was not completed at the time of this report.

Demonstratable Software. The only demonstration possible with the software developed in this project is the demonstration of the CCLNet operation with the Intel commercial networking software, installed in the first part of this thesis.

VII. Conclusions

Conclusions from this project are presented in three parts: the design summary, the problems encountered, and the recommendations for future study. The design summary is a quick overview of the contents of the Design section of this paper. The problems encountered is discussed in the next section. The concluding section is the Recommendations for Future Study, giving several follow-on efforts possible from this project.

Design Summary

The design of the communication software occurred in two phases: the TCP subset design, and the IP subset design. The TCP subset was structurally broken into two separate modules, one containing the user-accessible system calls for the software, and the other containing the processing procedures of the TCP subset. The system calls enter the control flow of the user process, whereas the processing procedures are a separate task started at runtime. The system calls communicate with the processing procedures through a set of common data variables accessible to both modules. A record, accessible by a system call, is kept of connection statistics during the time when the connection is valid. The IP subset design was structured into two procedures which process packets coming from and

going to the Ethernet. These procedures are passed control from the user process (the TCP subset kernel process, in this case). The Request Block is the main data structure used to communicate with the datalink layer of the Intel OpenNET software, and the parameter list on the procedure call supplies all of the data needed from the using process.

Problems Encountered

The major problems encountered in the process of pursuing this thesis effort involved the preparatory implementation of the CCLNet, the learning curve associated with the system, and the lack of documentation on the iNA961 request block field PORT_ID. The preparation included complete implementation of the commercial OpenNET system. This required much more time than was originally anticipated due to problems in acquiring the needed equipment and the learning curve of the author. When this objective was finally accomplished, the project had to be scaled down from a project to construct a gateway to a project to construct communication software with an eye towards a future gateway, again due to time constraints. A major problem involved locating pertinent information in the Intel operation manuals. The information is almost all there, but is not sometimes readily accessible unless an exhaustive search is conducted. This exhaustive search failed to turn up

information on what a programmer should use to determine what should go into the PORT_ID field of the request block.

Recommendations for Future Study

Several future efforts could be conducted from this study. These would include:

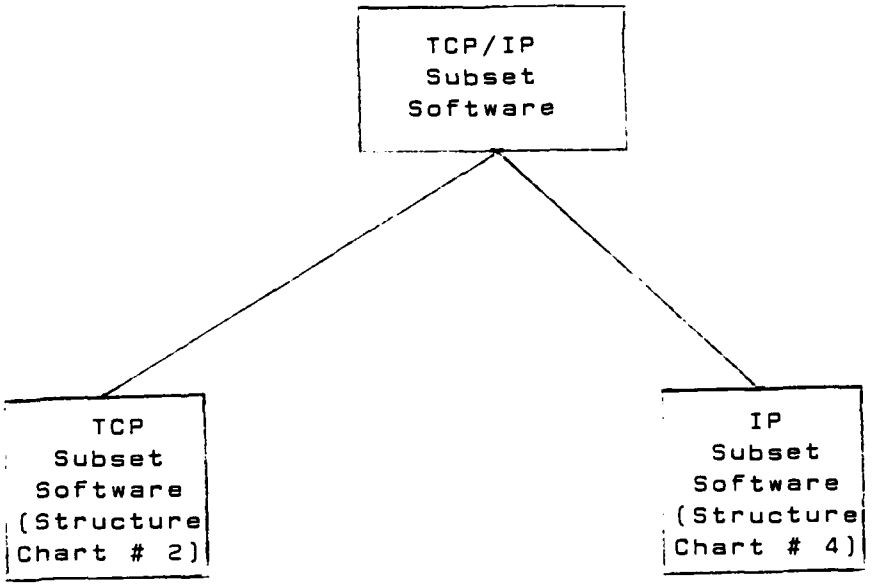
1. Future upgrading of the TCP/IP subsets to full TCP/IP implementations.

2. Construction of a gateway between the CCLNet and the AFIT TCP/IP Ethernet.

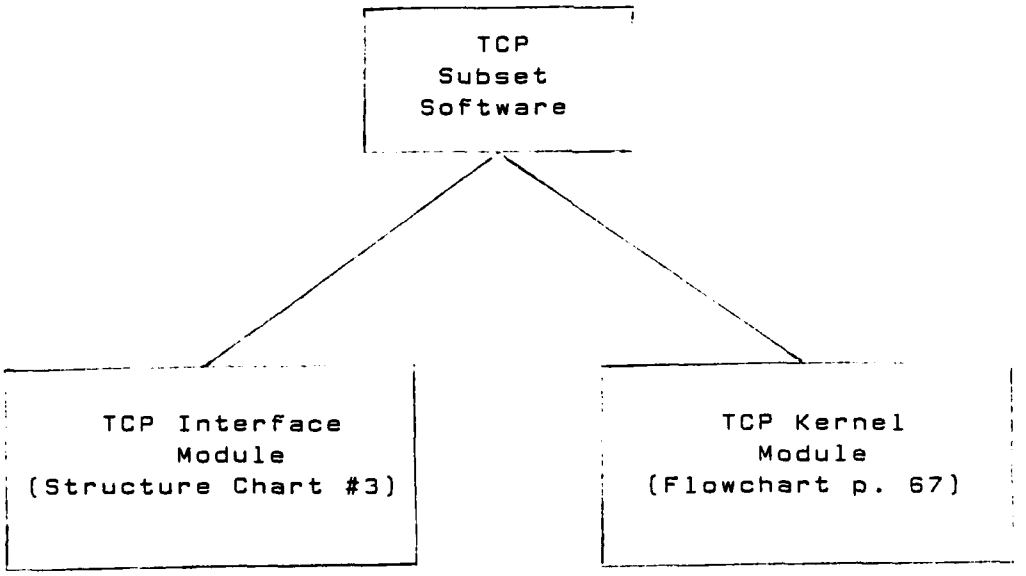
3. An in-depth analysis and evaluation of the CCLNet and potential uses and future improvements.

VIII. Appendices

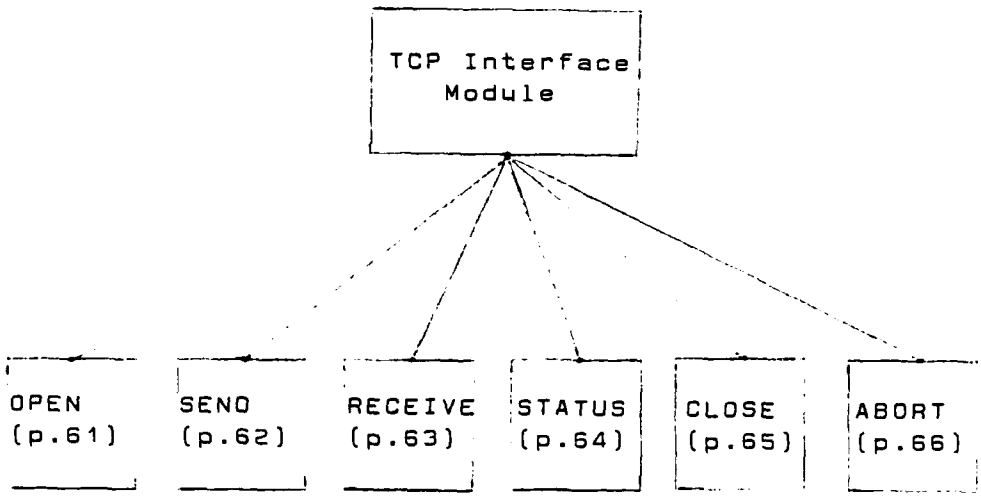
Structure Chart #1: TCP/IP Subset Software



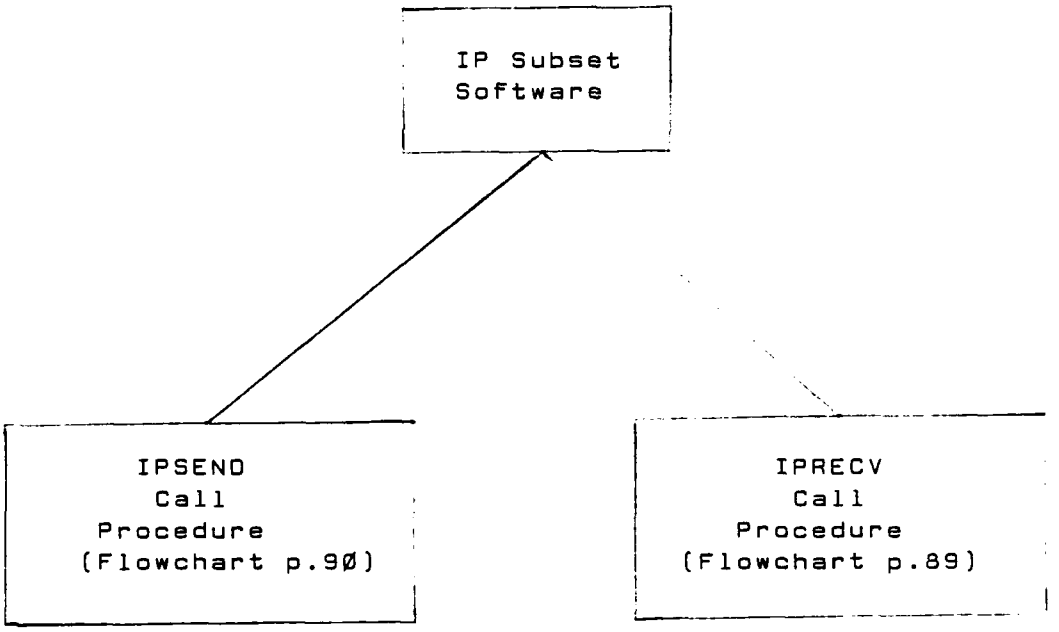
Strucure Chart #2: TCP Subset Software

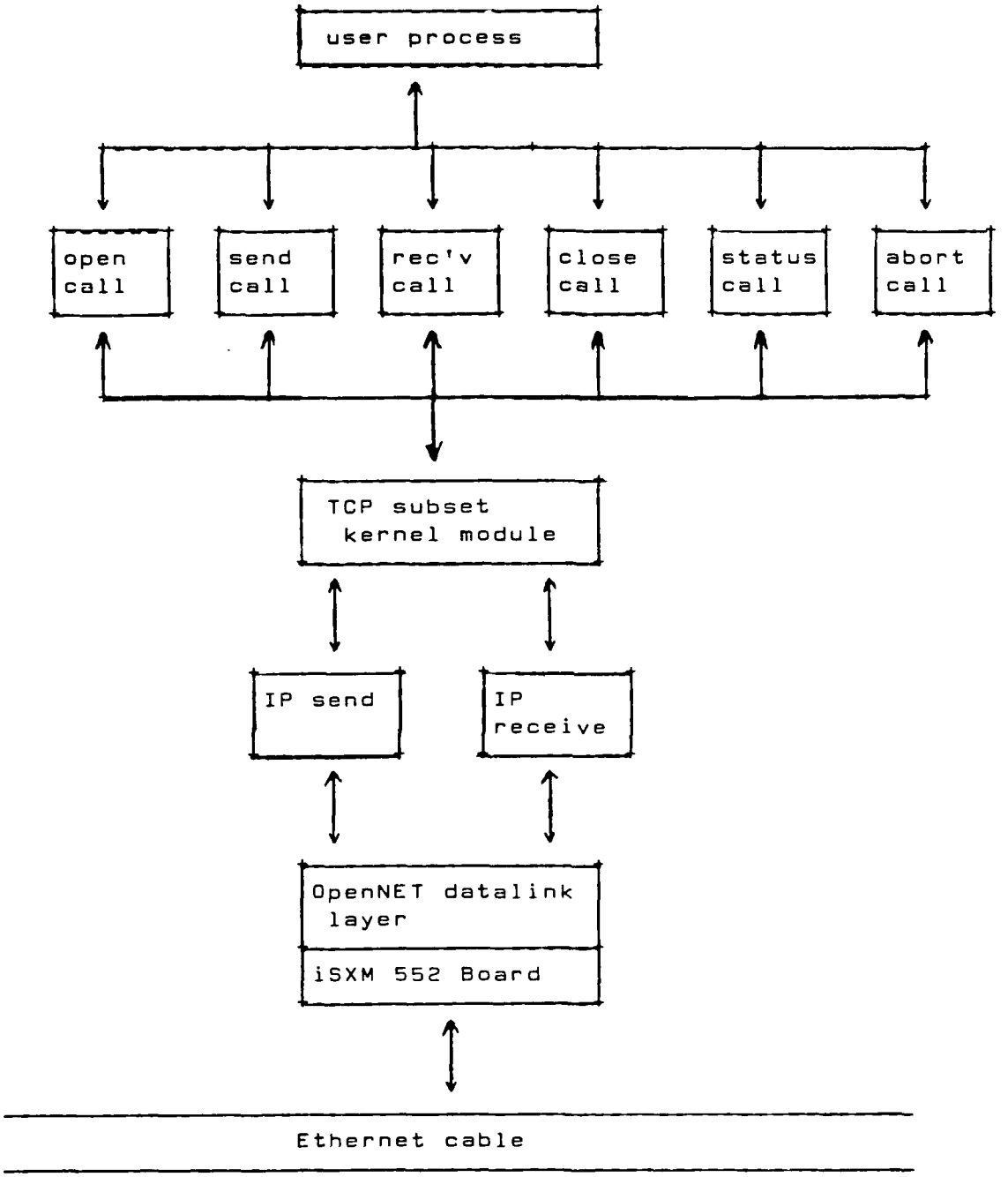


Structure Chart #3: TCP Interface Module

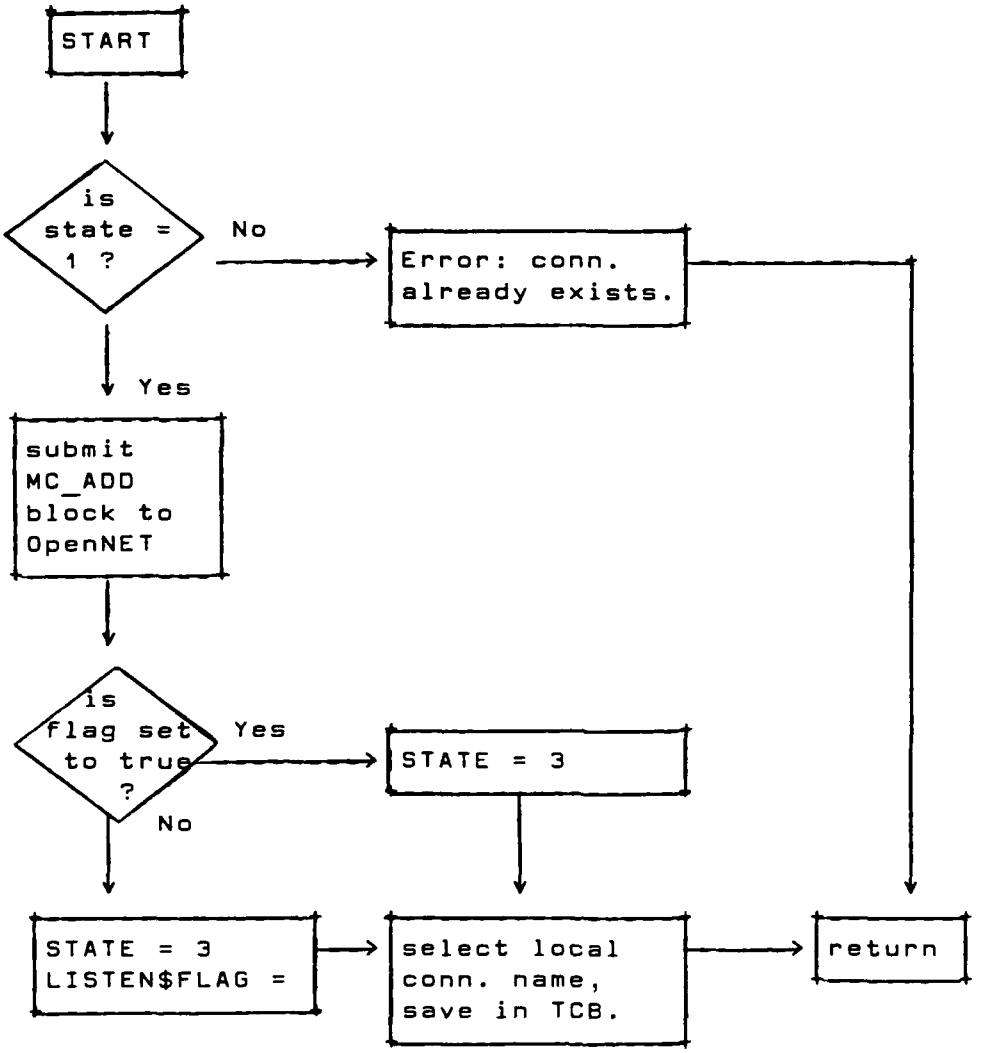


Structure Chart #4: IP Subset Software

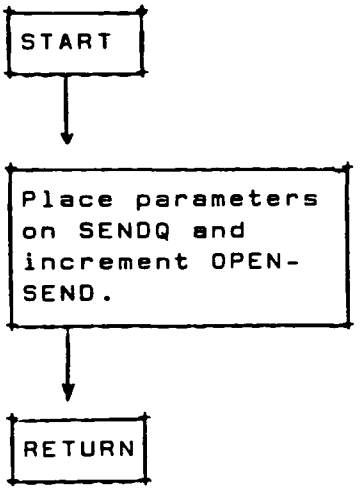



user process
open call
send call
rec'v call
close call
status call
abort call
TCP subset kernel module
IP send
IP receive
OpenNET datalink layer
iSXM 552 Board
Ethernet cable

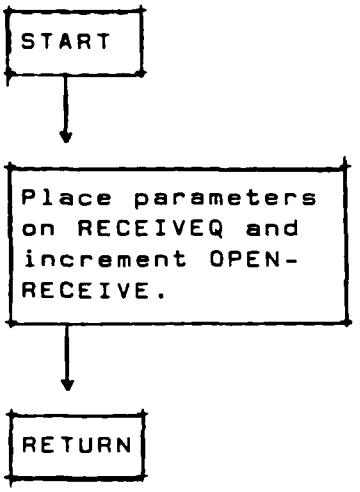
OPEN(local port, foreign socket, active-passive flag)



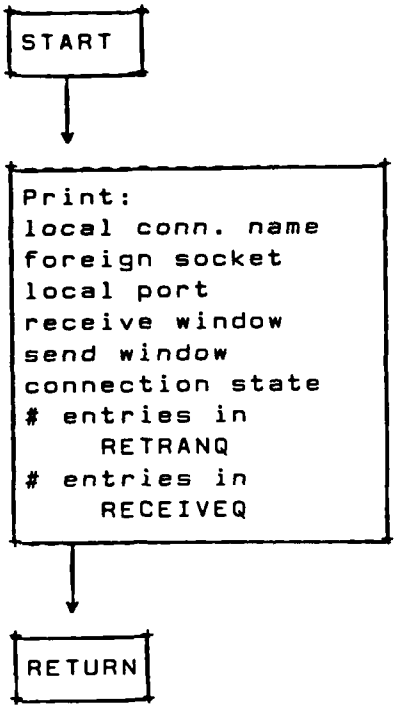
SEND(local connection name, buffer pointer, byte count,
EOL flag)



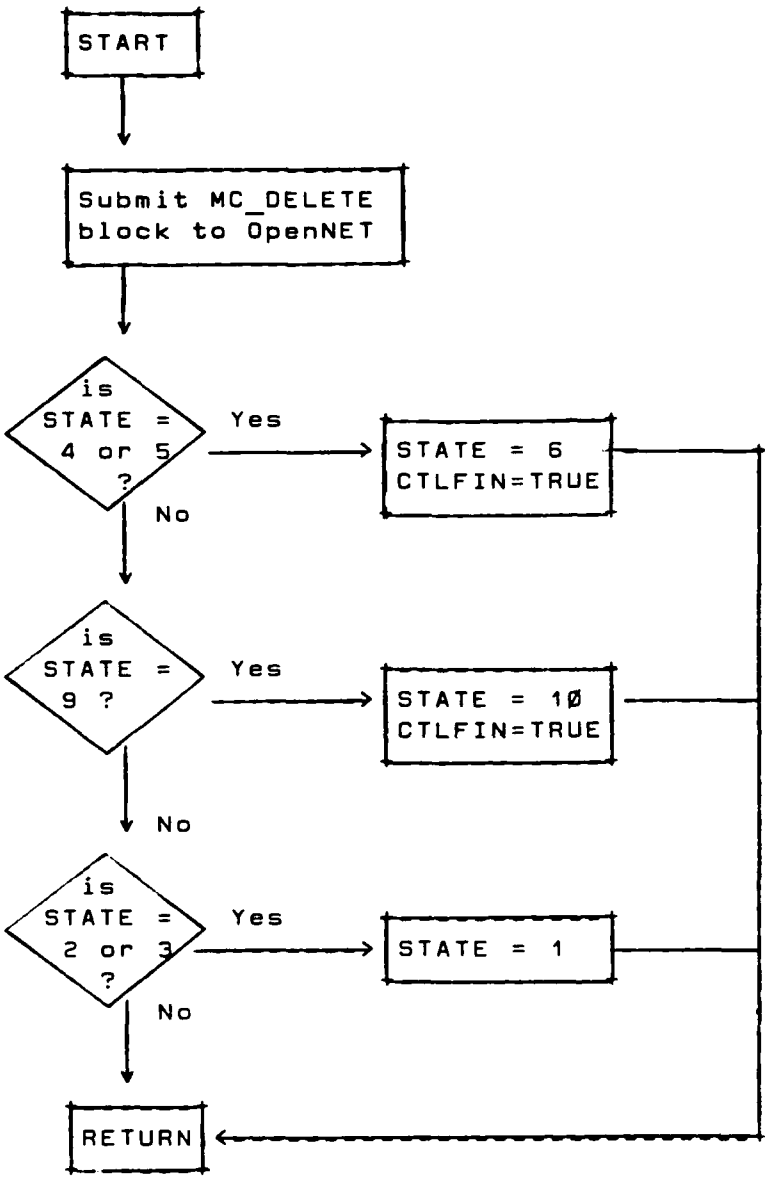
RECEIVE(local connection name, buffer pointer, byte count)



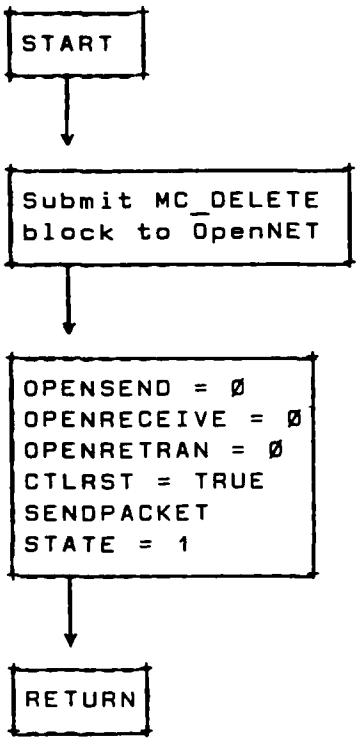
STATUS(local connection name)



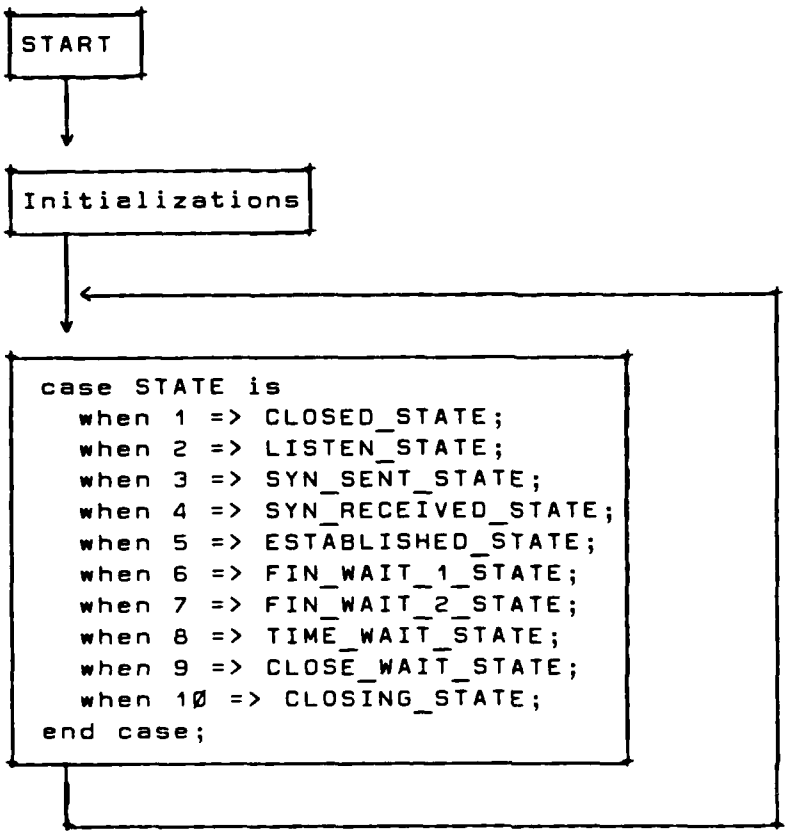
CLOSE(local connection name)



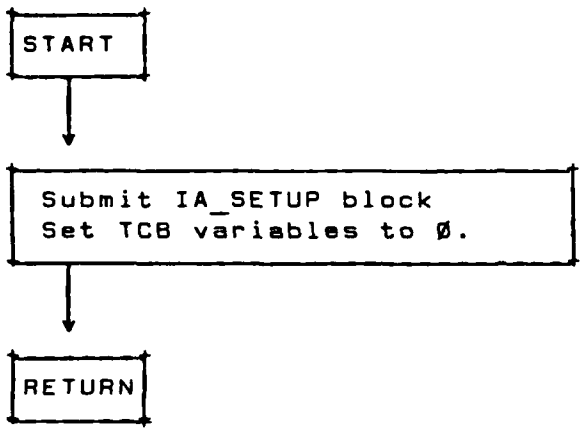
ABORT(local connection name)



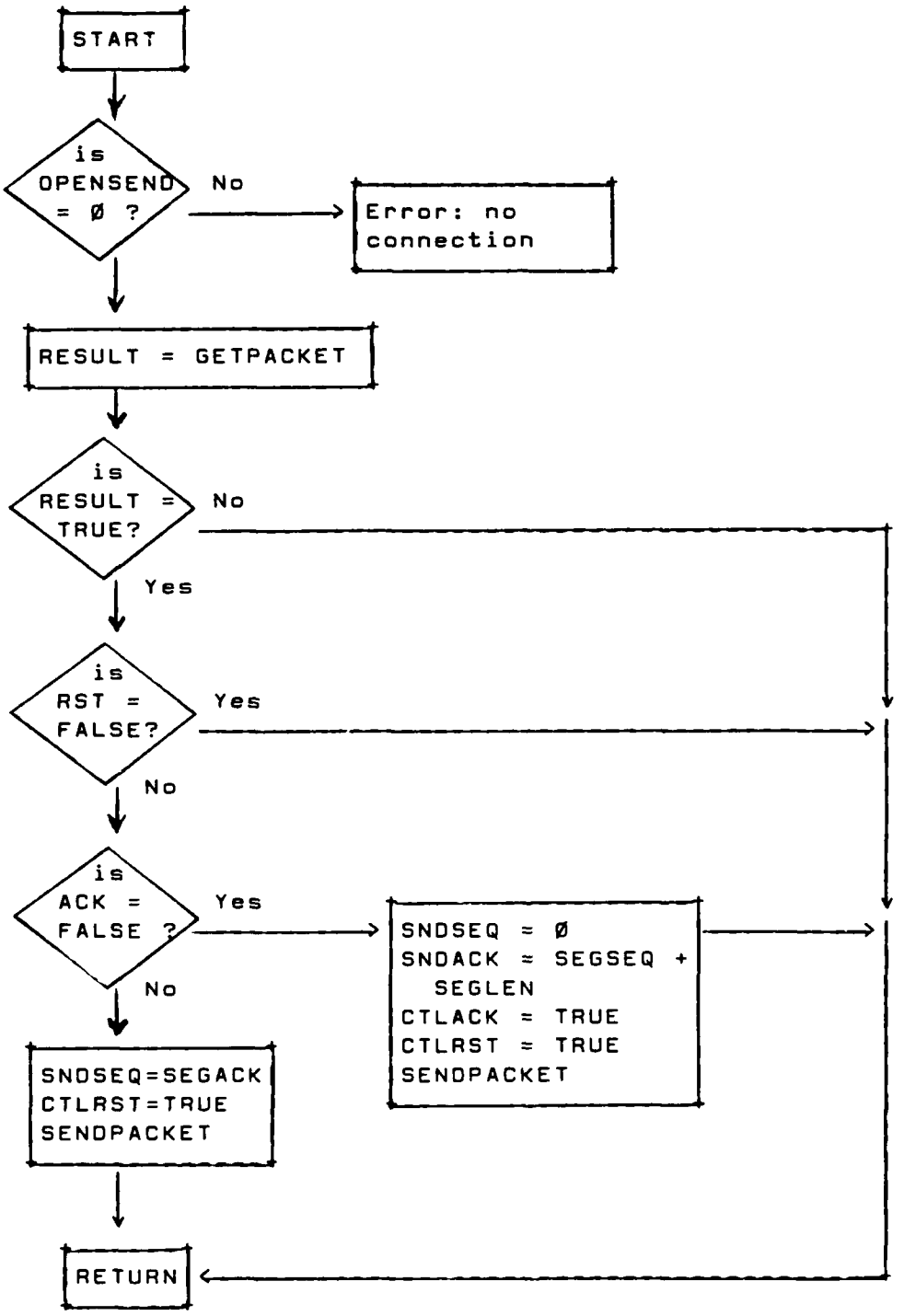
TCP Subset Kernel Module



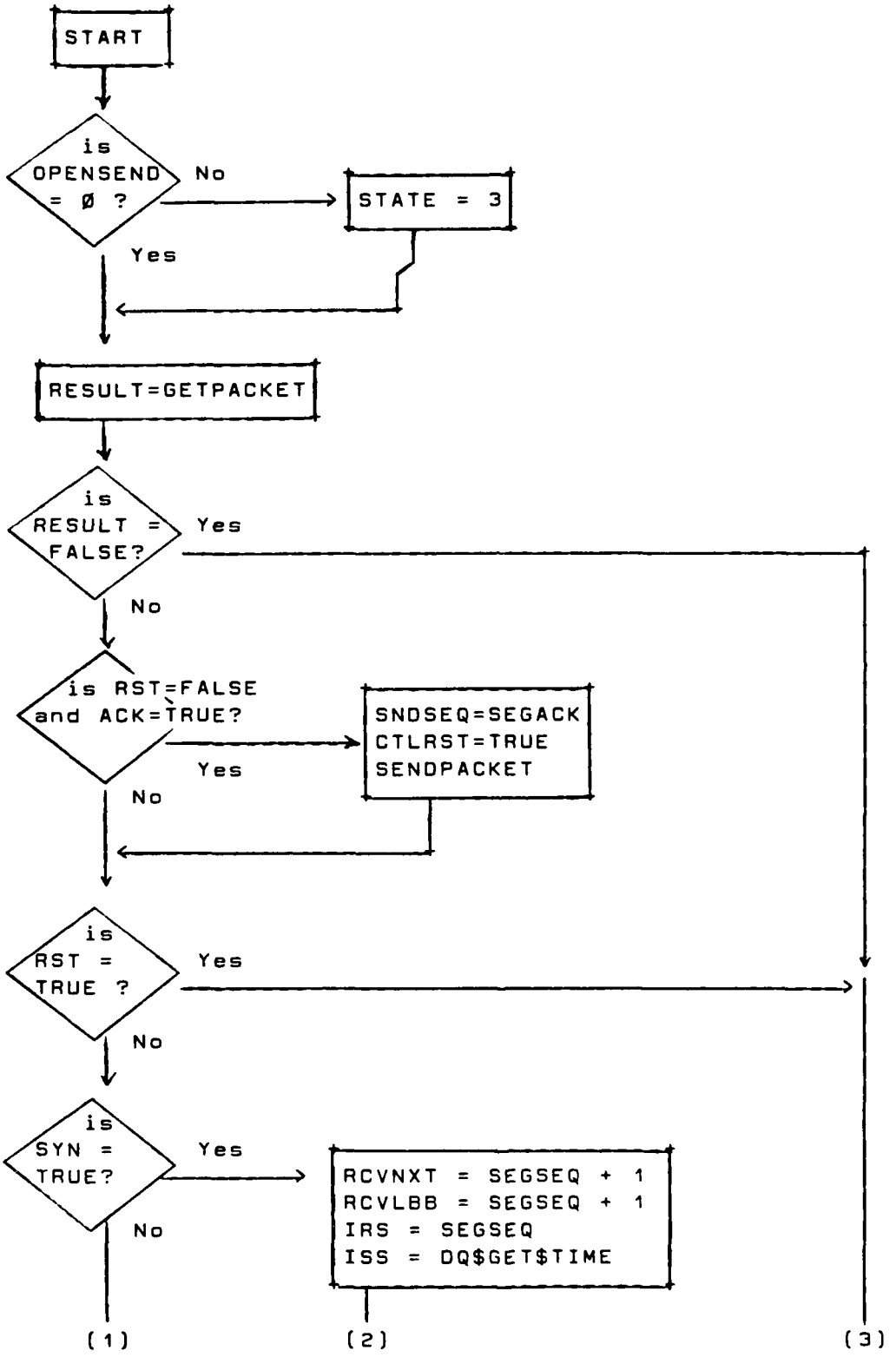
INITIALIZE



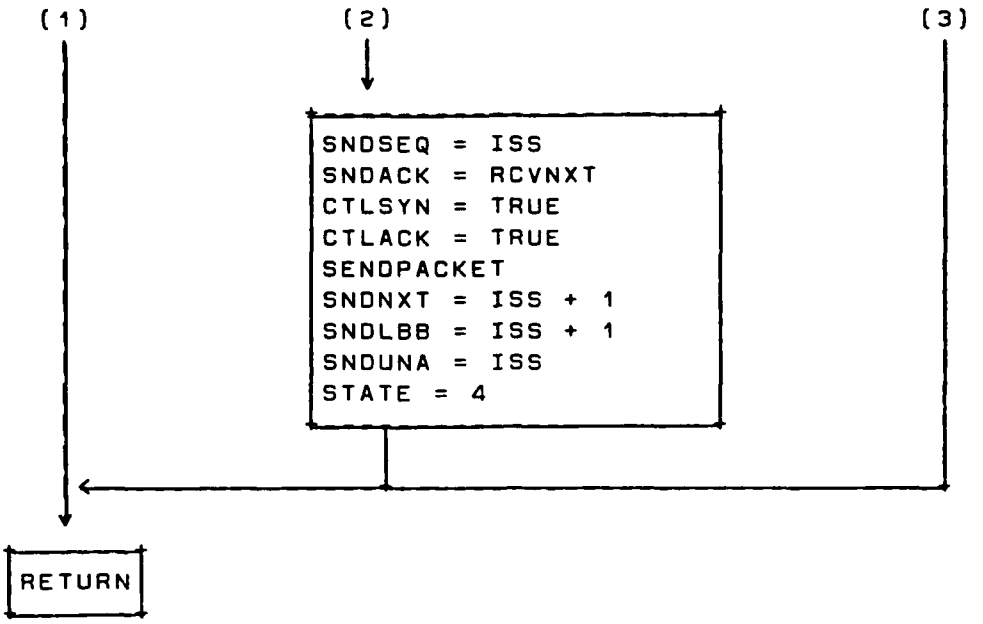
CLOSED_STATE



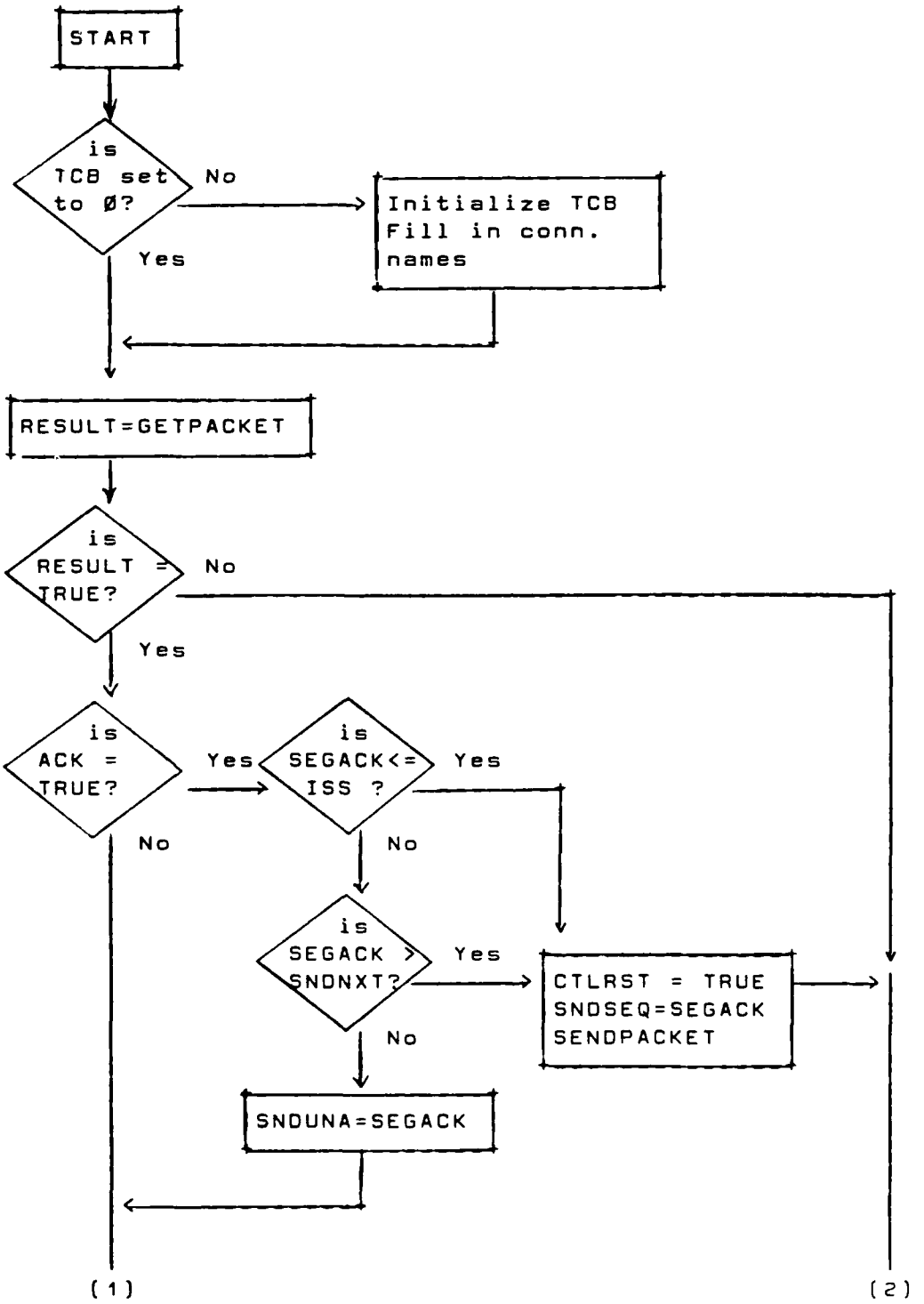
LISTEN_STATE



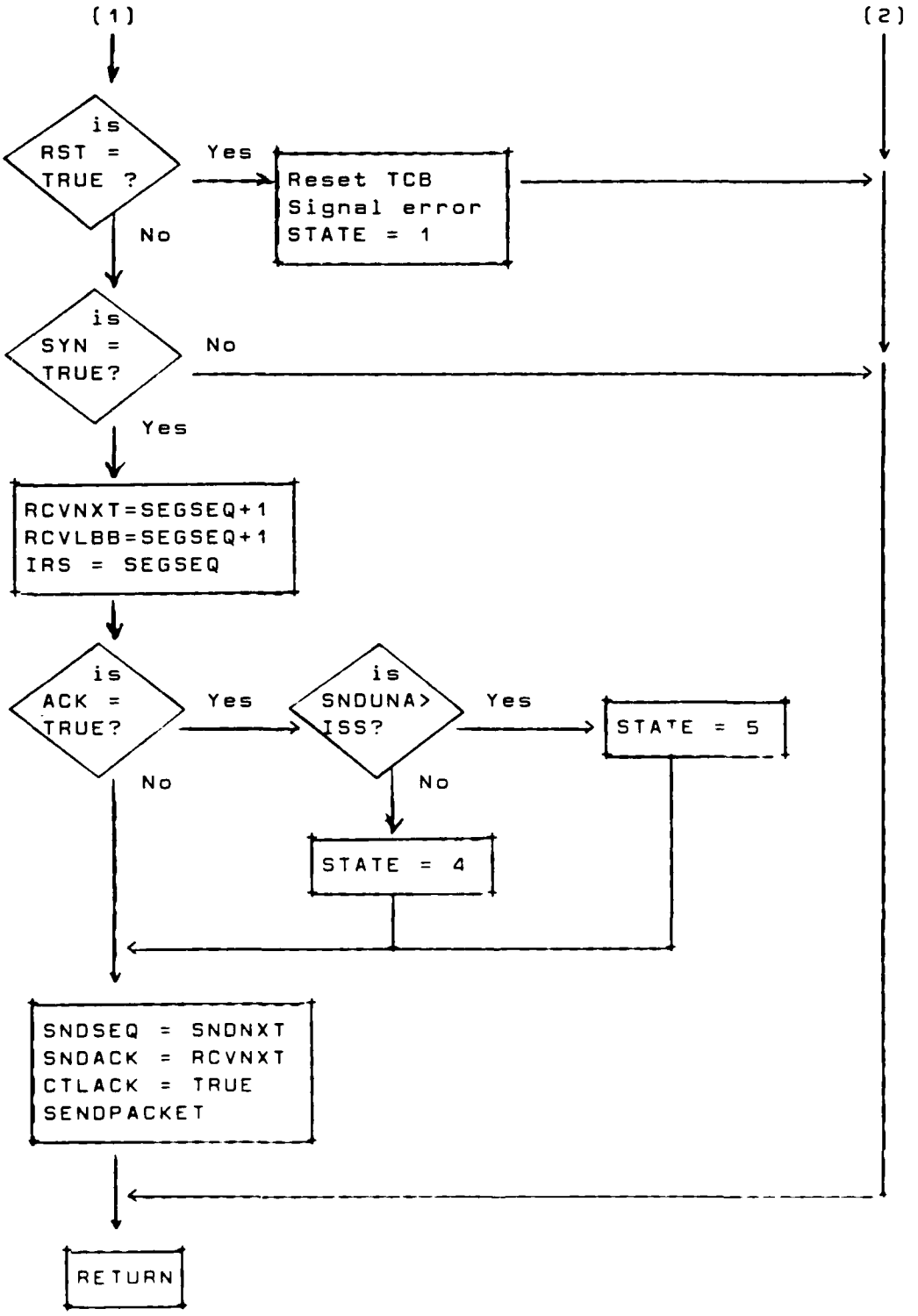
LISTEN_STATE (page 2)



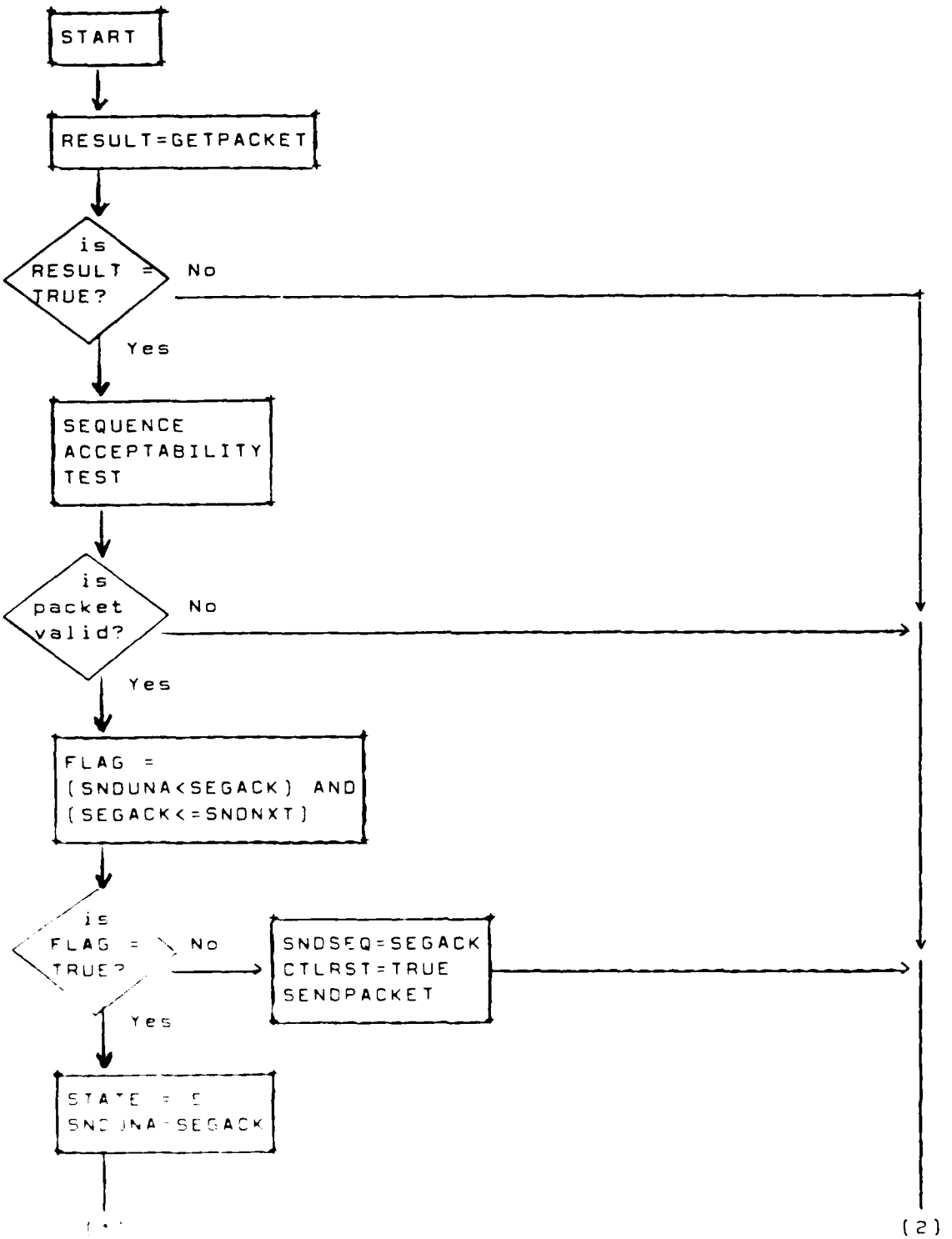
SYN_SENT_STATE



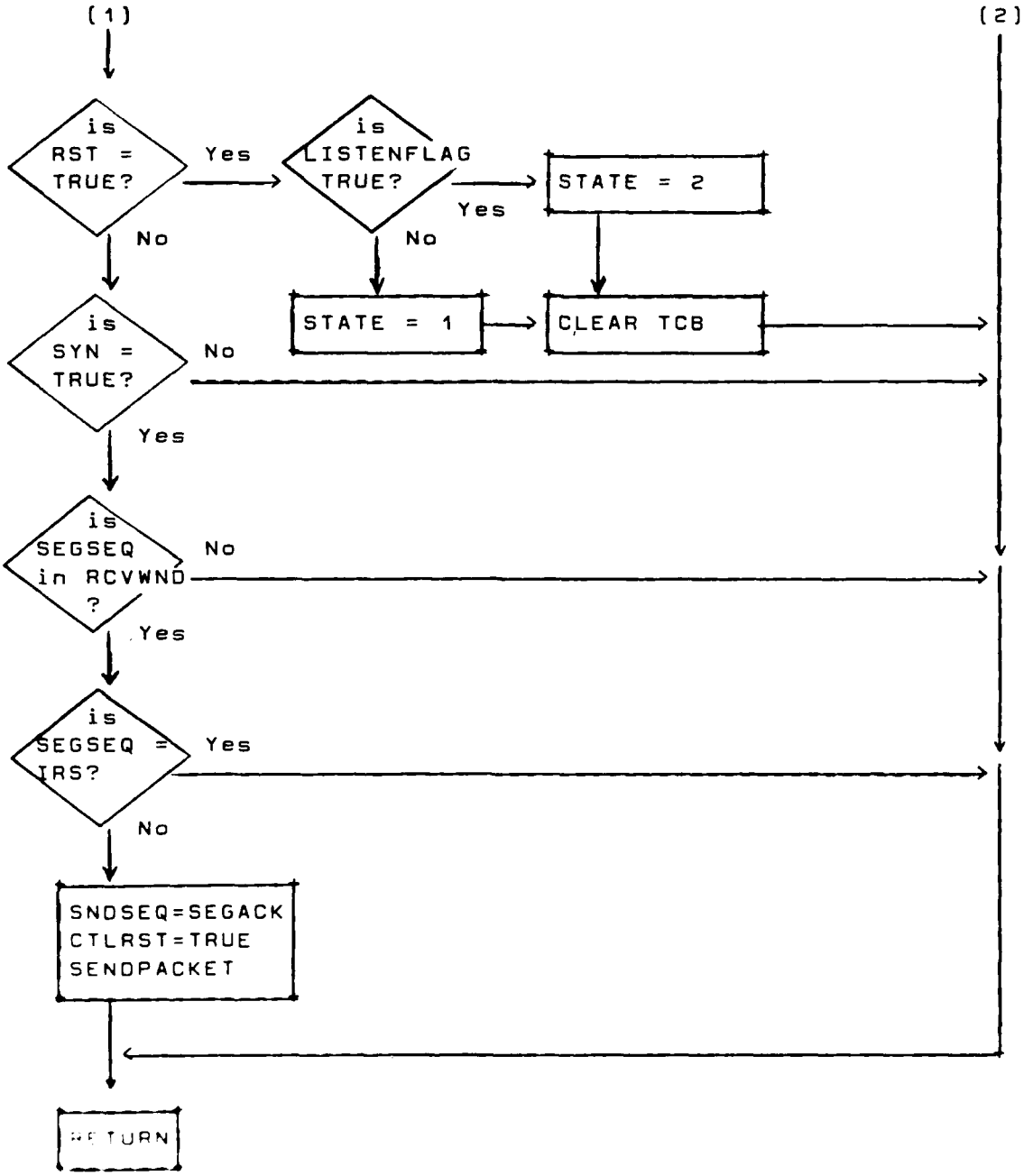
SYN_SENT_STATE(page 2)



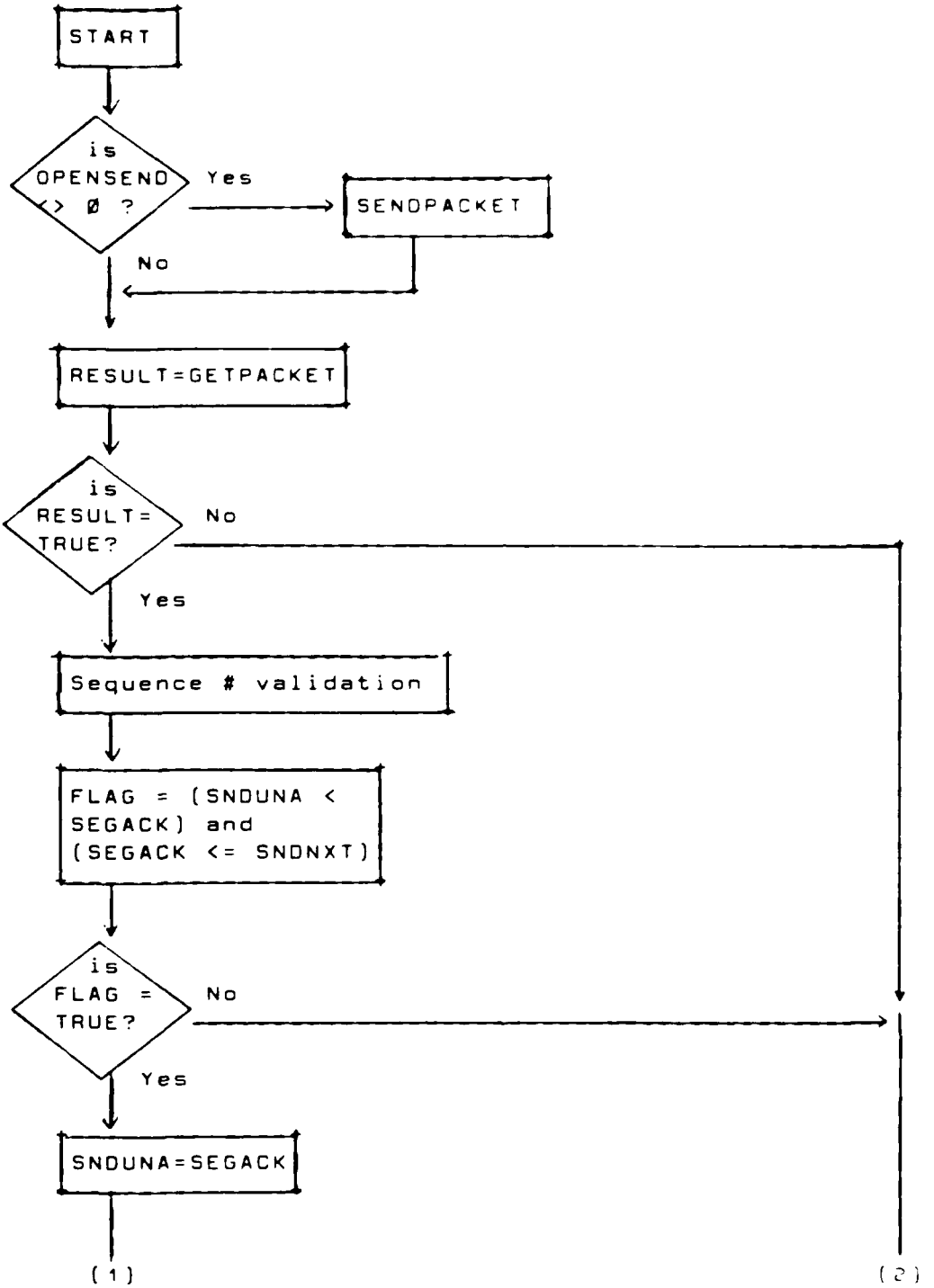
SYN_RECEIVED_STATE



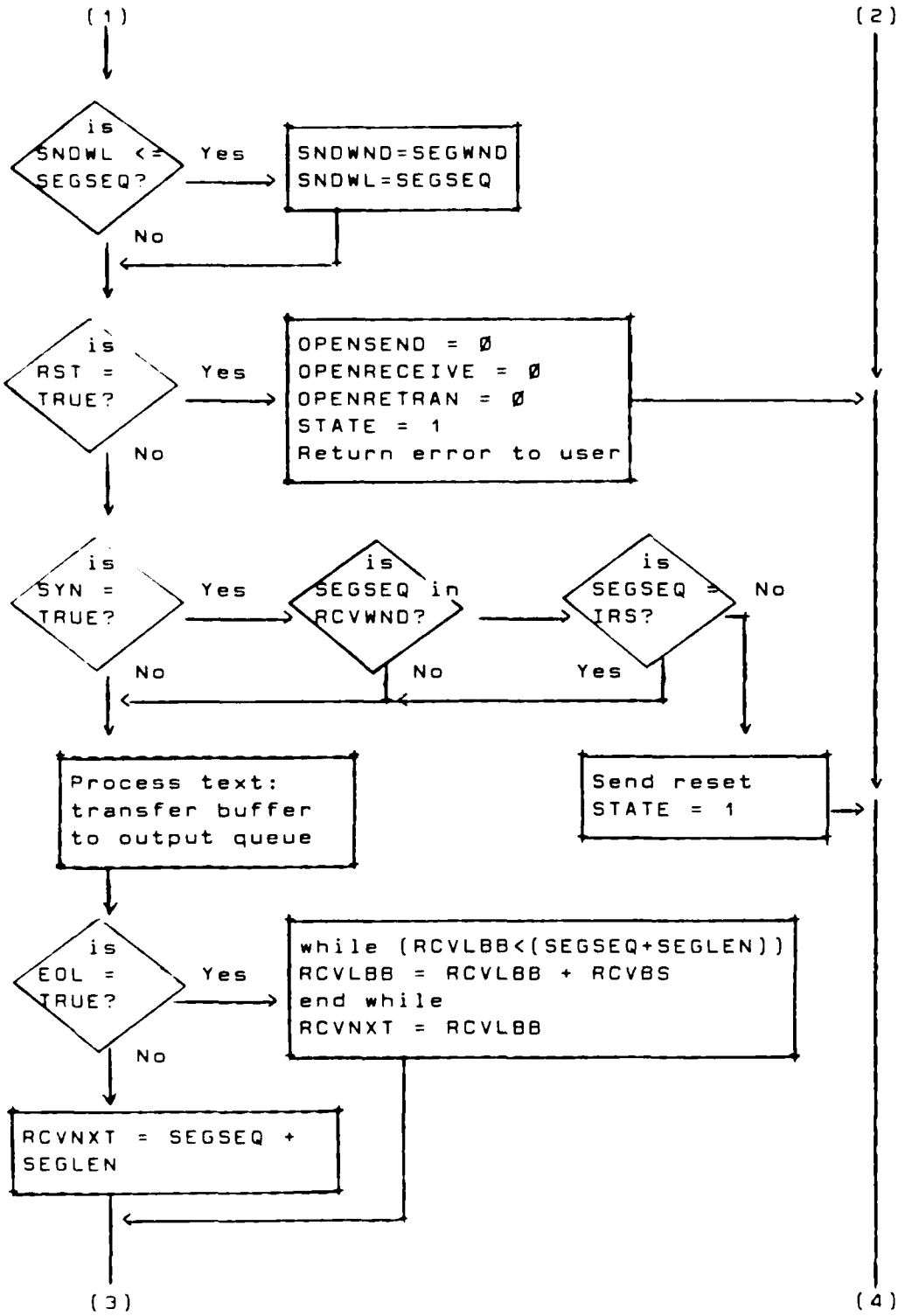
SYN_RECEIVED_STATE (page 2)



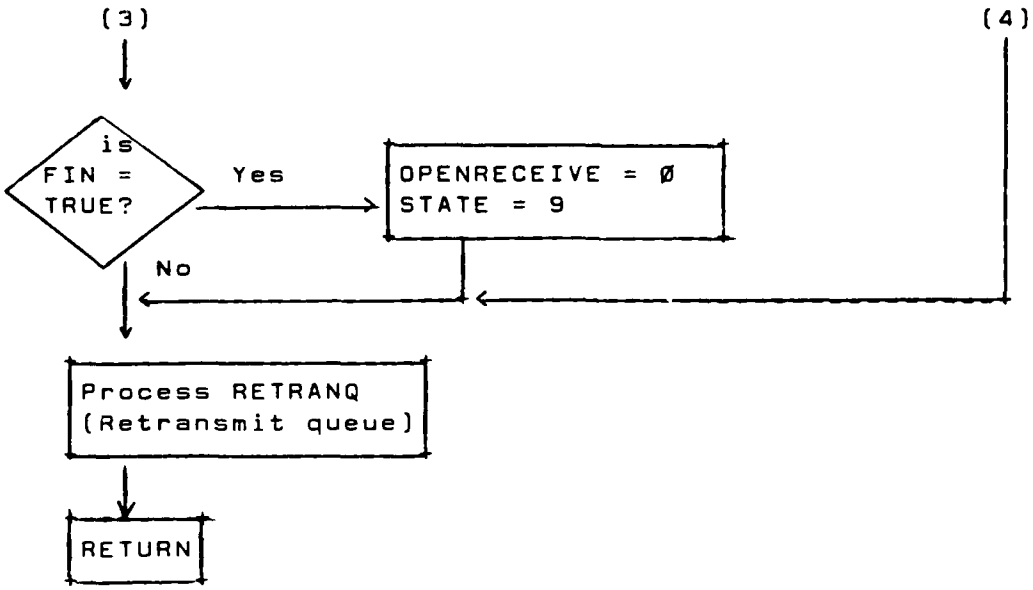
ESTABLISHED_STATE



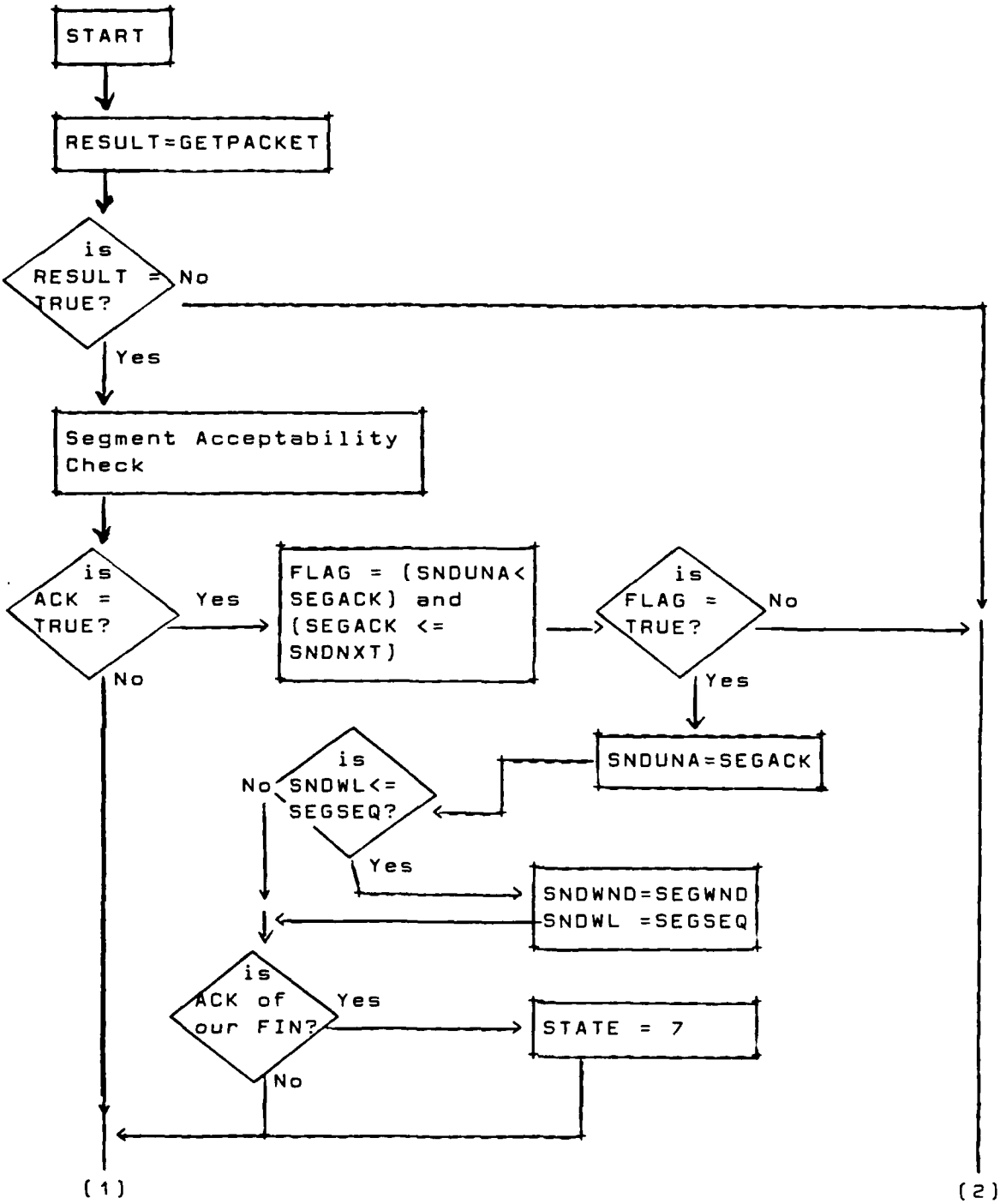
ESTABLISHED_STATE (page 2)



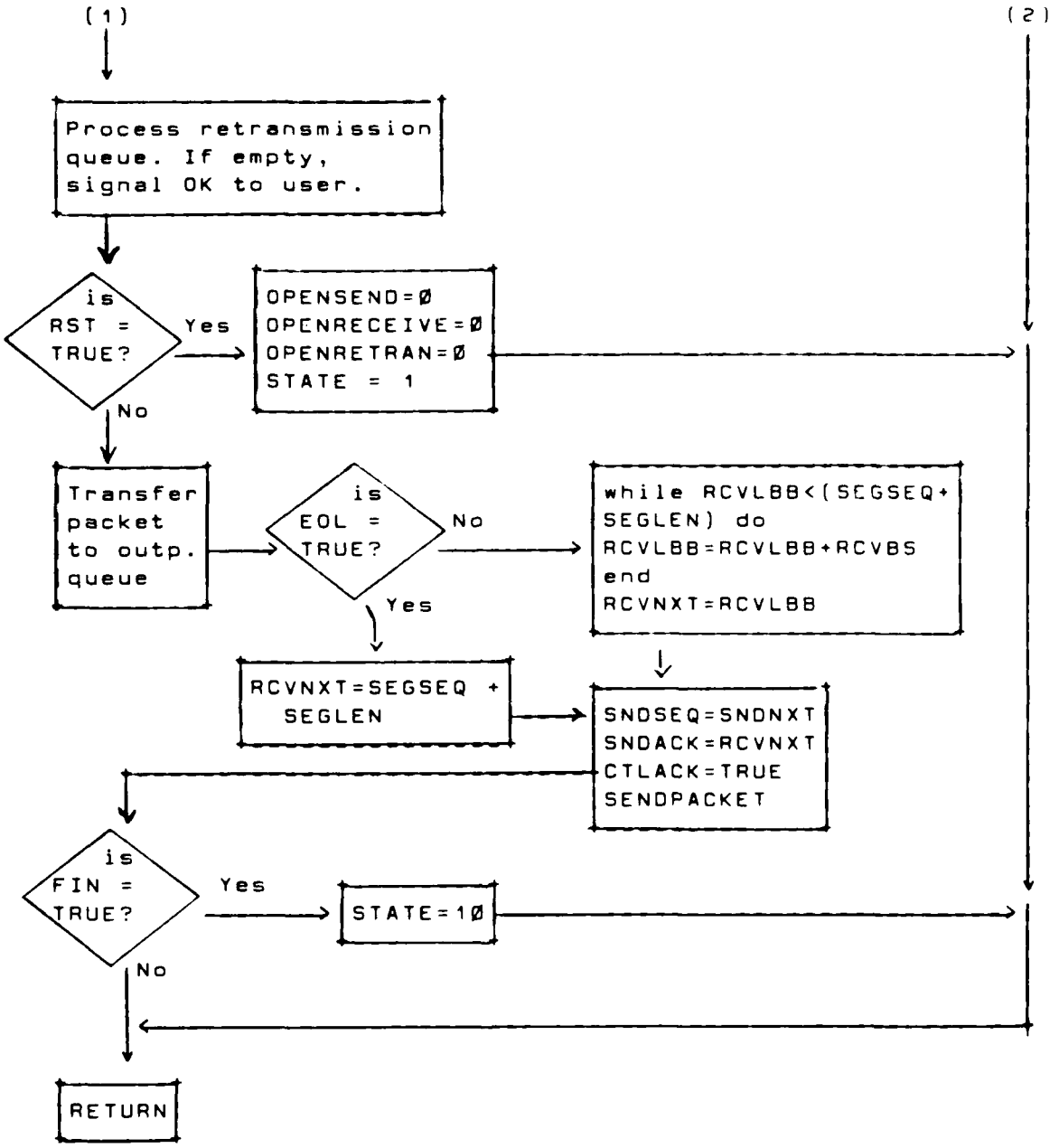
ESTABLISHED_STATE (page 3)



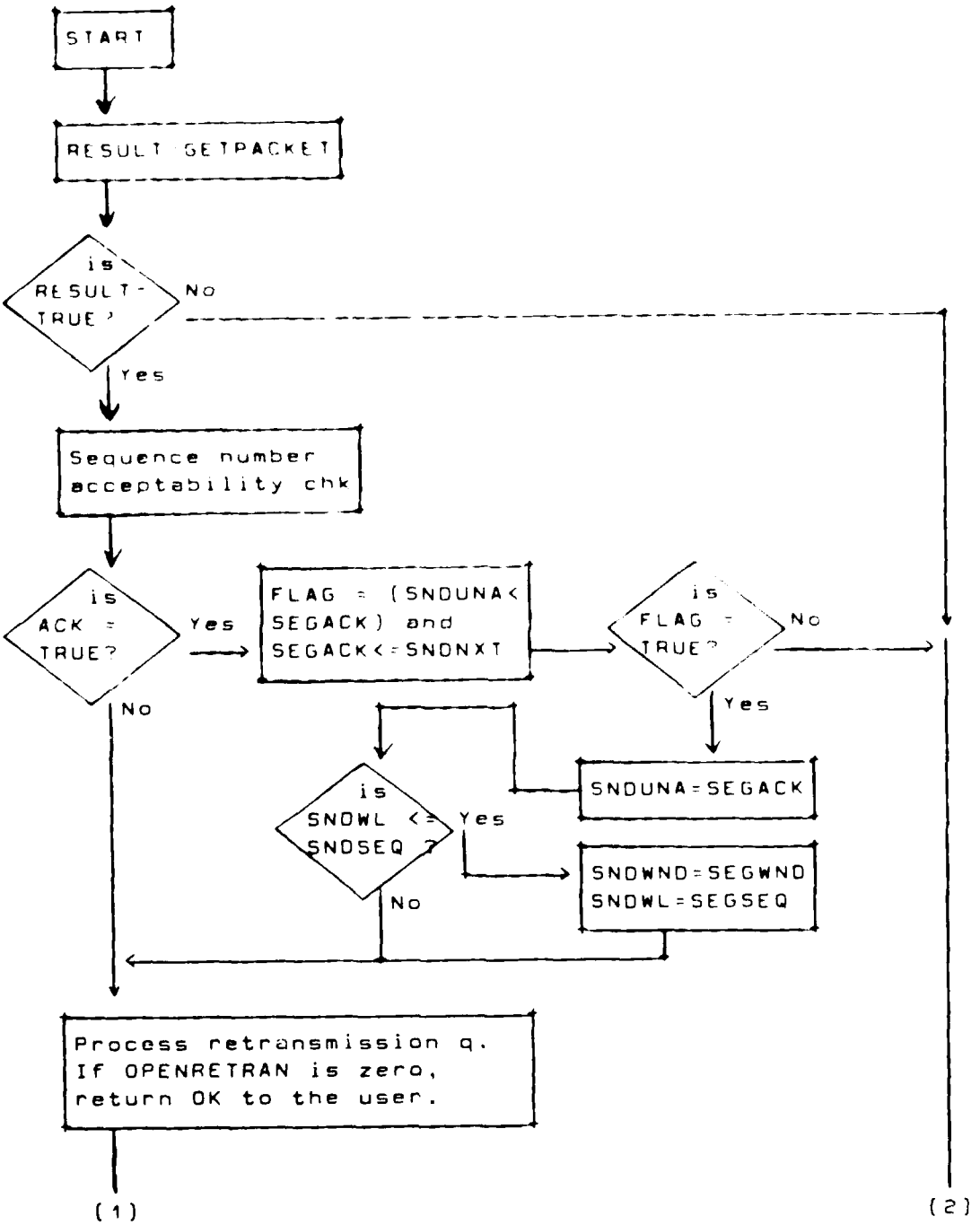
FIN_WAIT_1_STATE



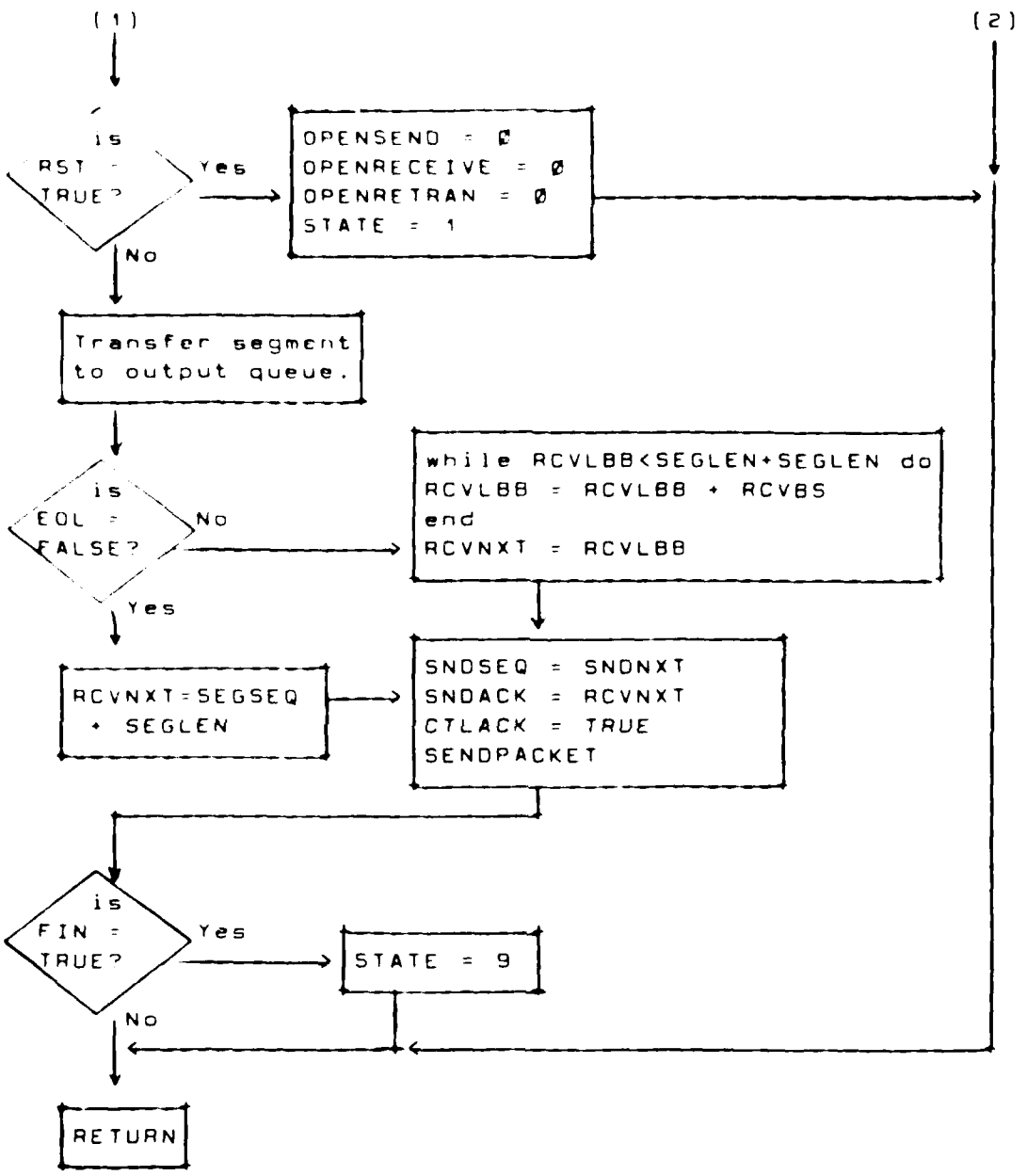
FIN_WAIT_1_STATE (page 2)



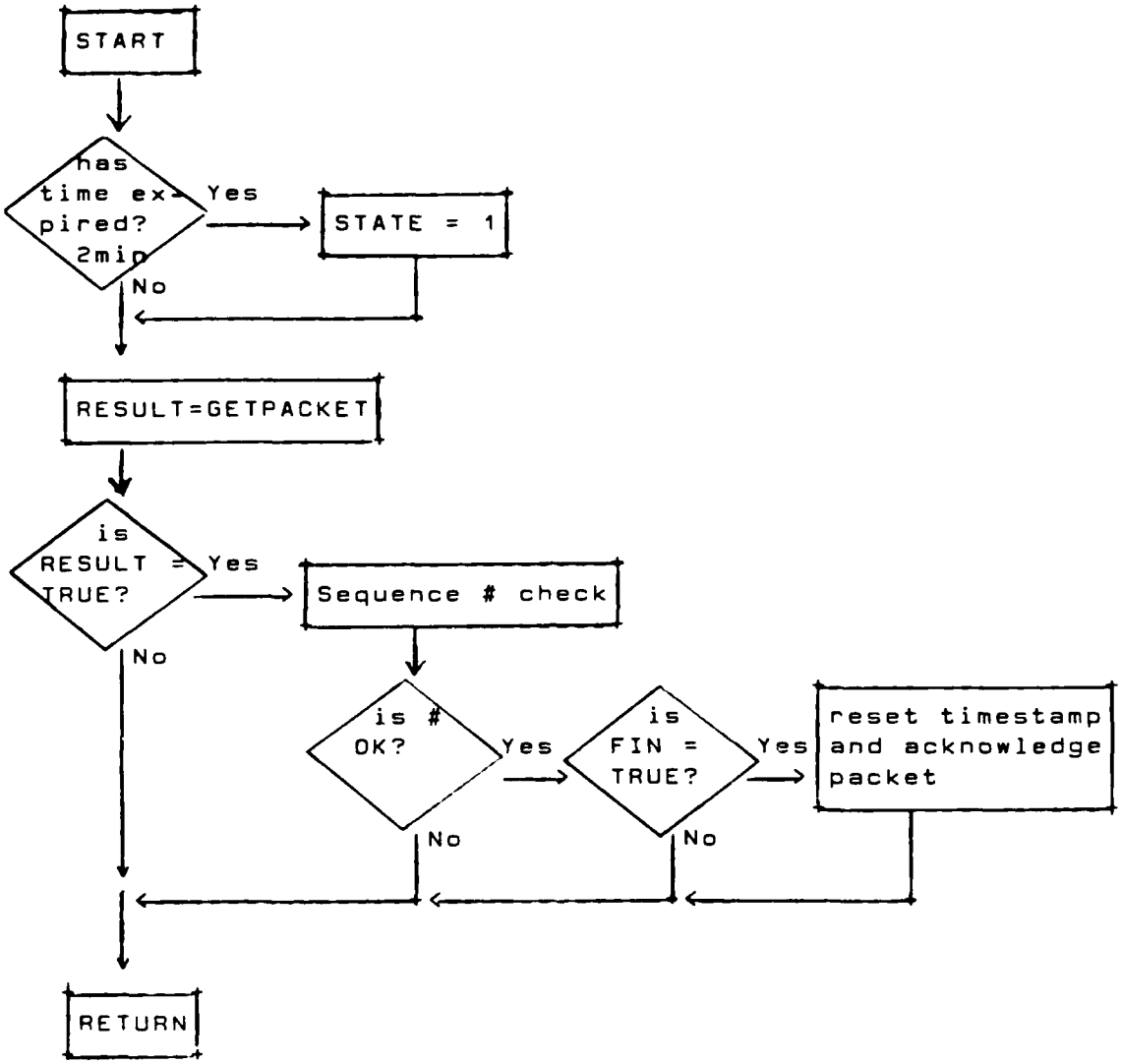
FIN_WAIT_2_STATE



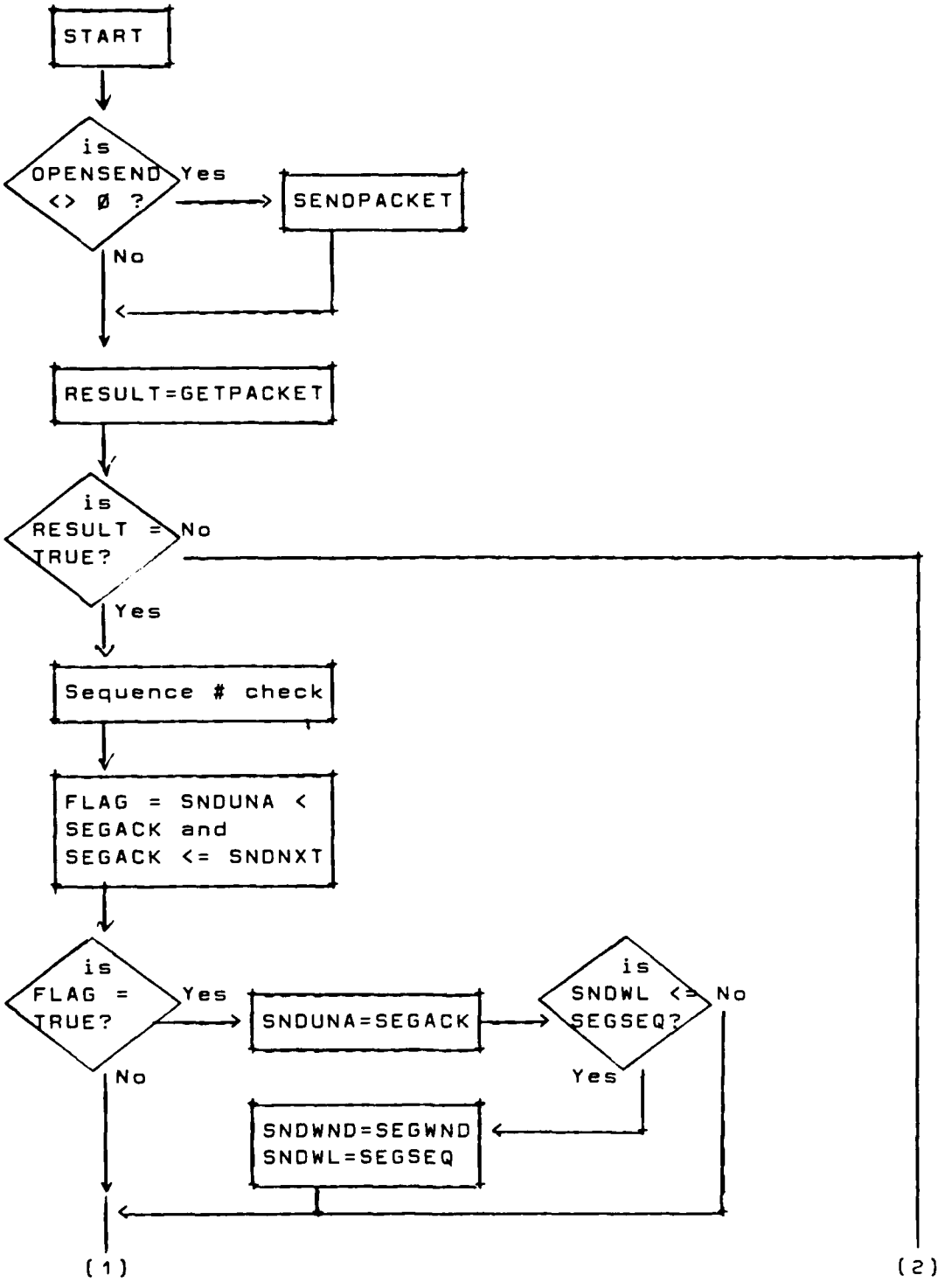
FIN_WAIT_2_STATE (page 2)



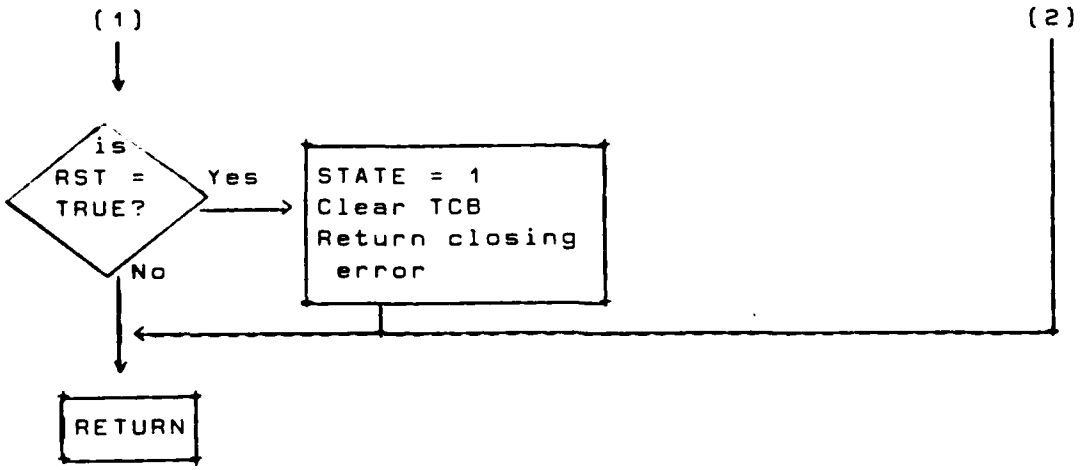
TIME_WAIT_STATE



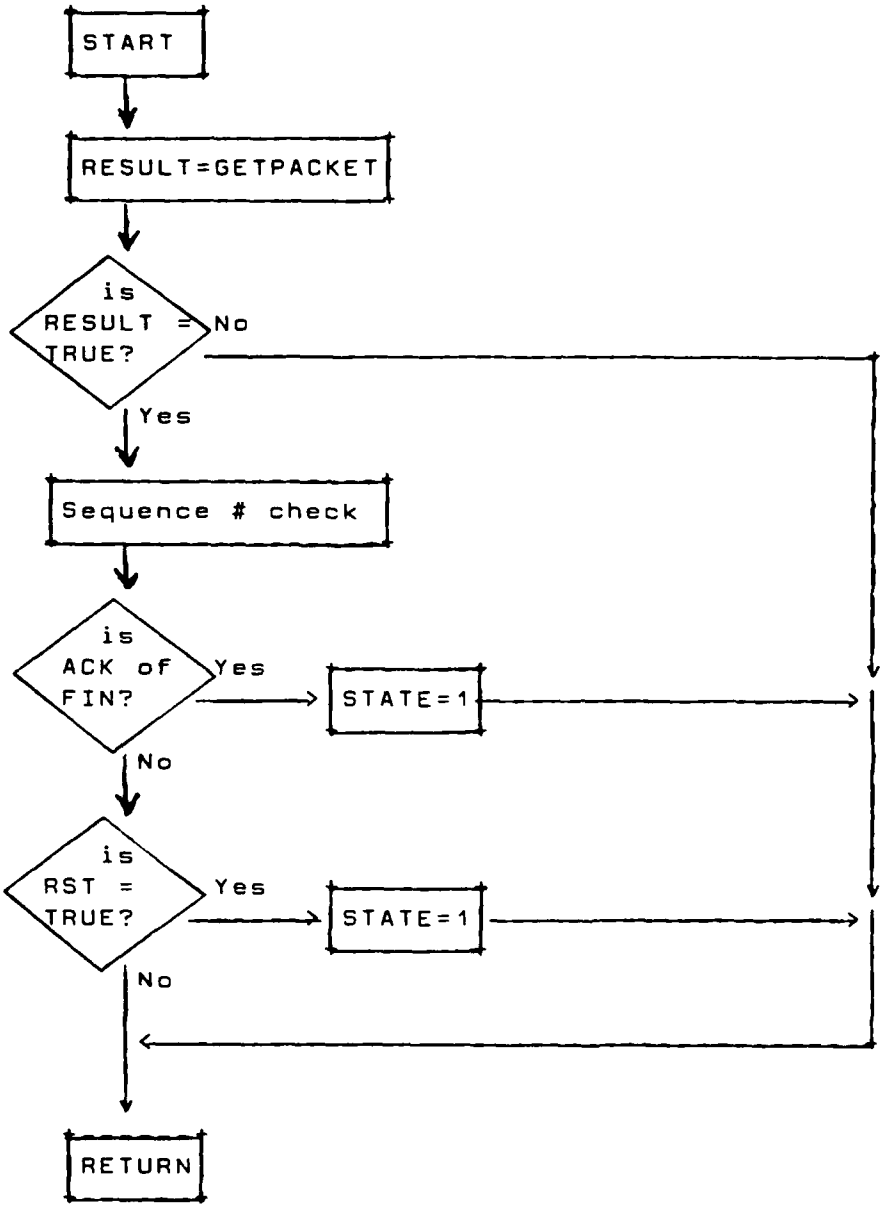
CLOSE_WAIT_STATE



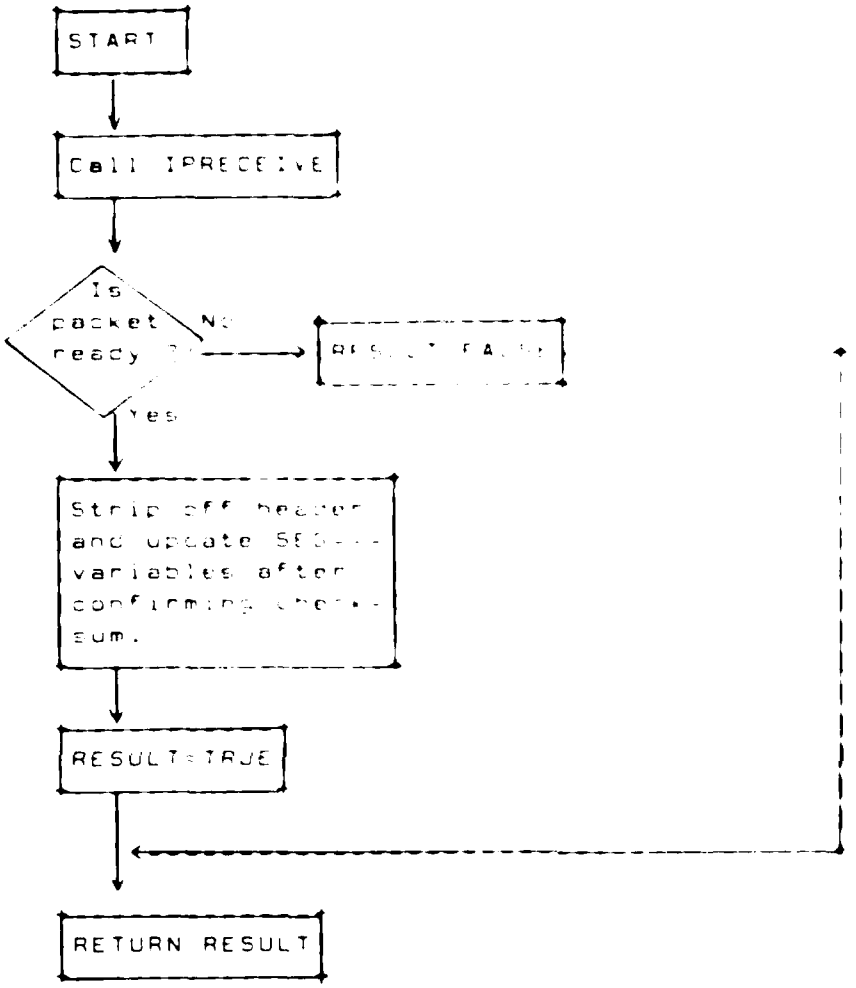
CLOSE_WAIT_STATE (page 2)

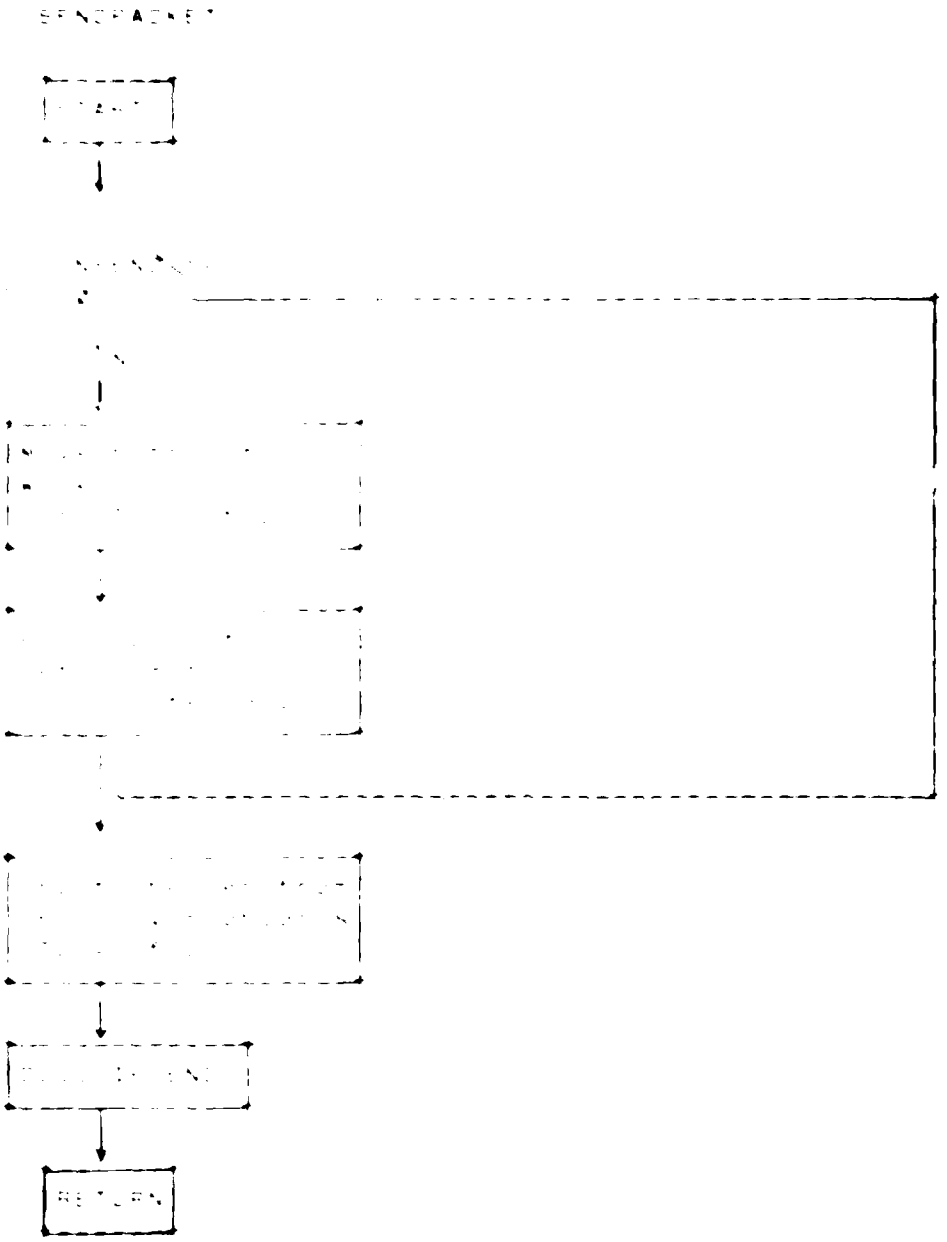


CLOSING_STATE



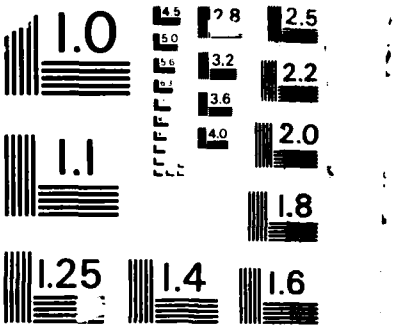
GETPACKET return BOOLEAN





AD-A177 606 DESIGN OF A TCP (TRANSMISSION CONTROL PROTOCOL) AND IP (INTERNET PROTOCOL..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. M B METZEL 2/2

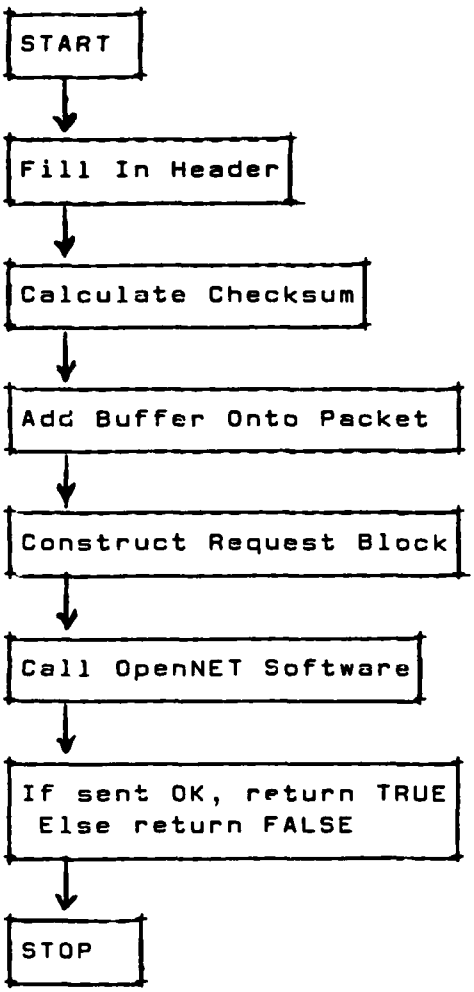
UNCLASSIFIED DEC 86 AFIT/GCE/ENG/86D-6 F/G 9/2 NL



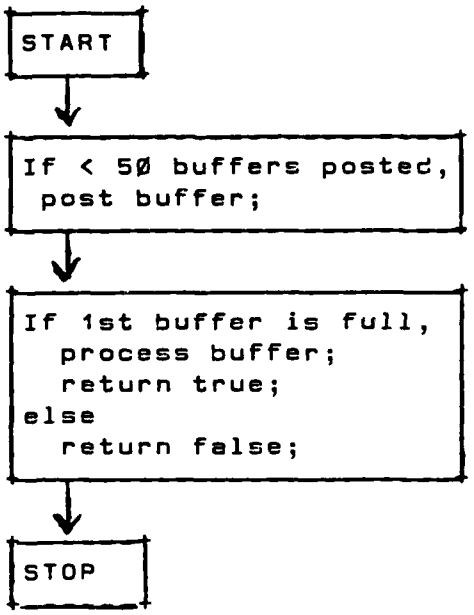
MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

IPSEND(Destination address, Type of Service, Time To Live, Buffer Pointer, Buffer Length, Identification, DF Flag, Options, Result);



IPRECV(Buffer pointer, Result, Source eddress, Destination Address, Protocol, Type of Service, Length);



IX. Bibliography

1. Blanc, Robert P. "Local Area Network Standards," IEEE Compcon Spring 1984 Digest of Papers: 252-254.

2. Booch, Grady. Software Engineering with Ada. Menlo Park, CA: Benjamin/Cummings Publishing Company, 1983.

3. Chlamtac, Imrich. "Exercising the Priority Mechanisms of the Intel 82586 Lan (Ethernet) Component in Support of Applications," Proceedings of the IEEE 1984 Computer Networking Symposium: 147-155 (December 1984).

4. Day, John D. and Hubert Zimmerman. "The OSI Reference Model," Proceedings of the IEEE, 71: 1334-1340 (December 1983).

5. "iNA 960 Programmer's Reference Manual." Santa Clara, CA: Intel Corporation, 1984.

6. "iNA 961 Programmer's Reference Manual." Santa Clara, CA: Intel Corporation, 1984.

7. "iRMX Networking Software User's Guide." Santa Clara, CA: Intel Corporation, 1985.

8. "iRMX86 Configuration Guide." Santa Clara, CA: Intel Corporation, 1984.

9. "iRMX86 Introduction and Operator's Reference Manual." Santa Clara, CA: Intel Corporation, 1984.

10. Mier, Edwin E. "Evolution of a Standard Ethernet," Byte, 9: 131-142 (December 1984). (Also as 6 in CCLNet section)

11. "Open Networking Strategy from Intel," Electronic Engineering, 57: 14 (April 1985). (Also as 7 in CCLNet section)

12. Padlipsky, M. A. "A Perspective On The ARPANET Reference Model." Proceedings of the IEEE Infocom '83 San Diego, CA 18-21 April, 1983 p. 242-253.

13. Postel, Jonathan B. (Ed.). "DARPA Internet Program Internet and Transmission Control Protocol Specification." Defense Advanced Research Projects Agency, Information Processing Techniques Office, RFC 790-796, DTIC ADA 111091, September 1981.

14. Postel, Jonathan B. (Ed.). "DoD Standard Internet Protocol." Defense Advanced Research Projects Agency, Information Processing Techniques Office, RFC 760, IEN 128, DTIC ADA 079730, January 1980.

15. Postel, Jonathan B. (Ed.). "DoD Standard Transmission Control Protocol." Defense Advanced Research Projects Agency, Information Processing Techniques Office, RFC 761, IEN 129, DTIC ADA 082609, January 1980.

16. Postel, Jonathan B. "Internetwork Protocol Approaches." IEEE Transactions on Communications, Vol Com-28, No. 4, April 1980, pp. 604-611.

17. Tanenbaum, Andrew S. Computer Networks. Englewood Cliffs, NJ: Prentice-Hall, Inc, 1981.

18. Weber, Capt. Mark W. "The Information Sciences Laboratory (ISL) Computer/Communications Laboratory Network." EE 650 Directed Study in Digital Systems. Air Force Institute of Technology, Wright Patterson AFB, OH, 6 December 1985. (Also as 8 in the CCLNet section)

VITA

Second Lieutenant Norman B. Hetzel was born on 20 April 1963 in Colorado Springs, Colorado. He graduated from Roy J. Wasson High School in 1981 and attended the University of Colorado in Boulder. In May 1985, he was awarded the Bachelor of Science degree in Chemical Engineering and was awarded a Reserve Commission as a Second Lieutenant in the United States Air Force upon graduation from the ROTC program as a Distinguished Graduate and Regular Commission selectee. He entered the Air Force Institute of Technology's School of Engineering in June, 1985, as a Computer Engineering major. He was commissioned as a Regular Air Force Officer in August, 1985.

Permanent address: 2432 Yorktown Road
Colorado Springs, CO
80907

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCE/ENG/86D-6	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) See box 19	
12. PERSONAL AUTHOR(S) Norman B. Hetzel, B.S., 2d Lt, USAF	
13a. TYPE OF REPORT MS Thesis 13b. TIME COVERED FROM _____ TO _____ 14. DATE OF REPORT (Year, Month, Day) 1986 December 15. PAGE COUNT 102	
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP
09	02	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Computer Communications

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: DESIGN OF A TCP AND IP SUBSET FOR INTEL 86/310 COMPUTERS

Thesis Chairman: Albert B. Garcia, Major, USA
Assistant Professor of Electrical Engineering

Approved for public release: IAW AFR 190-1.
WAYNE E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (AU)
Wright-Patterson AFB OH 45433

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Albert B. Garcia, Major, USA	22b. TELEPHONE (Include Area Code) (513) 255-3576 22c. OFFICE SYMBOL AFIT/ENG

DD Form 1473, JUN 86 *Previous editions are obsolete*

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Block 19 Continued

Abstract

As a precursor to a future gateway between the Air Force Institute of Technology (AFIT) Ethernet and the AFIT Computer Communications Laboratory Ethernet (CCLNet), the CCLNet was implemented using Intel Corporation iNA 961 and RMXNET software and then Transmission Control Protocol and Internet Protocol subsets were designed to be used on the Intel 86/31Ø RMX computer with OpenNET. Requirements definition, specification, design, and PL/M coding were used to solve the software design problem. The implementation and testing phases were not completed due to problems not covered in the system documentation.

END

4-87

DTIC